F5

Project Proposal

GraphQL in Enterprise Architecture

Updated May 1st, 2023

Colin Schutte
Gregory Hill
Riley Williams

F5 Mentors:
Maxwell Wynter
Rajesh Narayanan

TABLE OF CONTENTS

1. INTRODUCTION

The ability to defend against bots is an important and key issue threatening many companies today, as it has become a persistent and dangerous threat to the security of many companies' servers and services. According to www.signalsciences.com, bot attacks include any automated web request with a fraudulent goal involved and can have a negative impact on everything from websites, applications, and APIs to the end-users. Over time, these bot attacks have transformed from small spamming operations into complex, multinational criminal infrastructures (1). This is becoming ever more important as AI technologies are becoming exponentially better.

TMS is a test management suite offered by F5 that provides a customer with the ability to test endpoints or clients of their own systems and see if they are human or bots. In this project, we will be working with a group of mentors from F5 to evolve their product in a positive way through the migration of databases and the integration of a GraphQL API.

The overarching goal of this project has been to investigate GraphQL architectures vs. RESTful architectures. The TMS system that we are working in has a 3-tiered architecture where the UI communicates with a REST API in order to retrieve the data from the tabular database. This architecture is traditional and very popular but has some issues that we want to solve through an exploration of GraphQl and the evolution of the system to a GraphQL architecture.

In order to reach this goal, we will need to migrate data from the current database used by the TMS stack to a graph database and then create and integrate a GraphQL API that will query this graph database. The reasons as to why this needs to happen start with the problems faced while using their current database. To begin, they face a "JSON blob problem" which stems from the tabular nature of the database. In the tabular SQL database used, there is data stored as JSON(JavaScript Object Notation) objects which are extremely expensive to parse and handle due to them being in the form of trees and graphical in nature. These objects store important data gathered from the issuing of tests for consumers and can be used by F5 and their consumers to improve their products and security. Using JSON is a popular and efficient format to transfer data between servers and web applications and is heavily used throughout the web space today, so there is great importance in keeping these blobs in their proper format and using them to their full potential. Migrating these JSON blobs to a directed graph database will allow for more efficient, less expensive parsing, sending, and receiving, and allows this data to have the directed relationships of which it was intended.

An example of one of these JSON objects is one that stores a test and the data about the test such as its test version. Right now, TMS doesn't store this data so that it can be used effectively. If a user wanted to see the transformations of a test from version to version, they wouldn't be able to as it is stored in a JSON object. If we were to move this data to a graph database, we could store each test and quickly query for different versions of tests and see the modifications.

Along with solving this issue with JSON objects, migrating this data to a graph database will also solve problems that developers face when trying to add functionality to this system. Due to the nature of this data being graphical, it makes it easier for developers when trying to add functionality. An example of an added functionality could be the ability of uploading whole programs. Programs are also graphical in nature and F5 plans on taking advantage of this with the ability to parse these programs to improve their services. This functionality could be used to parse programs such as the policies that F5 uses as tests and instead of having to parse these programs in the tabular database, they will already be parsed and ready to be queried.

To conclude, integration of a graph database to TMS would not only solve the JSON blob problem, but also allow the F5 team to add functionality to this system in a way that would not be possible if the data were still in the current tabular format. It has become ever so important to make improvements on a system that provides clients with defense against bot attacks, and the migration to a graph database will ensure this.

2. BACKGROUND

Our project includes many different aspects, and because of this a wide variety of background knowledge is required. Our project is best understood from the beginning and working forward. That is, starting with JSON objects and working forward to the overall structure of the project, using GraphQL and Neo4j to create and handle a graph database.

Javascript Object Notation (JSON)

An understanding of JSON objects and how to model them will be crucial at the beginning of the project when generating our GraphQL schema. JSON is an open standard file format used to contain and transfer human readable data using key-value pairs. An example of a simple JSON object that stores information about a pet cat would look like this:

```
{
      "name": "Baloo",
      "age": 10,
      "species": "Main Coon mix",
      "owner" : {
            "name": "Riley",
            "age": 33
      }
}
```

This JSON object contains simple key-value pairs and shows that the values can be of any type, including another JSON object. The first key-value pair has the key “name” and is followed by a colon and then the value “Baloo” to represent the name of a pet cat. In this case, the value of the key is stored as a String. The second key-value pair, representing the cat's age, has the key “age” and is followed by the value 10 illustrating that the type of value to be stored is an int. The last key-value pair has the key “owner” followed by the value of another JSON object, which contains simple information about the owner.

The object above illustrates the human readability of the JSON format. It is easy to discern the characteristics and meaning of the cat object.

A more exact standard of JSON can be found at ECMA-404 (2).

JSON → GraphQL Schema

Once an understanding of JSON objects has been obtained, it is important to be able to abstract the objects to a schema, or a representational model, of the JSON object. That is, given a specific case of a JSON object, create a model that could be used to represent any JSON object of a similar type. This abstraction will be used for other tools which will be explained further on. From our cat example above, instead of a concrete JSON object representing a specific cat named Baloo we would create an abstraction that would be able to model any pet cat. A schema that models a pet cat would look like:

```
type Cat {
    name: String!
    age: Int
    species: String
    owner: [Owner!]!
}

type Owner {
    name: String!
    age: Int
}
```

The schema has many similarities to the example JSON object. Instead of concrete values like we saw in the JSON object, the schema contains the type of value expected for that key. An important difference from the JSON example is that we need a representation of Owner as well as Cat. Without the definition type for Owner, the owner field in cat wouldn't have a valid data type.
Another important feature of schemas is the ability to make fields required, or non-nullable, when creating types. An example can be seen in both the Cat and Owner types for the name field, where the return type is String!. The exclamation point at the end of String means that the name field can not be empty, there must be a string there. Another more in depth example is the owner field in Cat. Here the return type is [Owner!]!, which represents a non-null list of non-null Owner elements.

Below is an illuminating example of how this works with a String array:

```
exampleField: [String!]

exampleField1: null         Valid
exampleField2: [ ]          Valid
exampleField3: [a, null, c] Invalid
exampleField4: [a, b, c]    Valid
```

Above examples 1, 2, and 4 are all valid forms of exampleField as they fulfill the requirements of having no null elements in the string array. It might seem like example 1 wouldn't be a valid example, but because the array is allowed to be null it doesn't invalidate the required type for the field.

```
exampleField: [String]!
```

```
exampleField1: null        Invalid
exampleField2: [ ]         Valid
exampleField3: [a, null, c]Valid
exampleField4: [a, b, c]   Valid
```

In this example 2, 3, and 4 are all valid types. Example 1 is not valid because exampleField must have a non-null list as a return type, even if it doesn't require the elements of the list to exist.

If we now consider the owner field of the example Cat type we can see that only examples 2 and 4 would fulfill the requirements because the list is not null, and the elements in the list are not null.

In contrast the age field of cat can return null because it does not have an "!" after Int. This is to represent a situation where the cat's age is unknown. A more comprehensive schema might include an Age type in addition to Cat and Owner to accurately represent an unknown age.

GraphQL Schema → Neo4j Node (GraphQL OGM and typeORM)

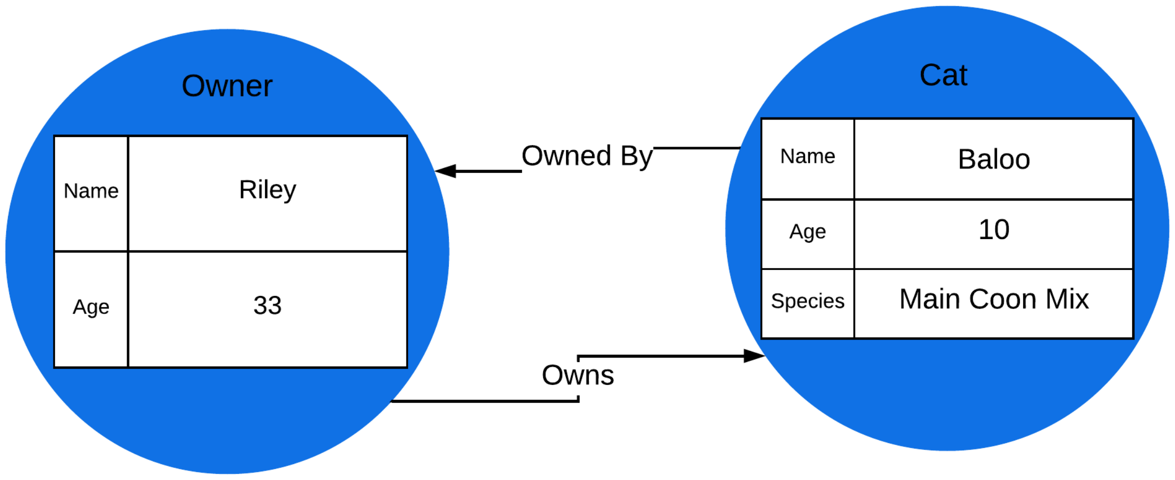
Now that we have schemas to work with, we can begin to use them along with GraphQL and typeORM to take data from the relational database and store it as nodes in the graph database. TypeORM is an open source ORM (object-relational mapping) made for typescript projects that we used to pull data from a SQL database and map it to a desired object. We started by creating these objects for all of the entities in the MySQL database where we also add all of the entity-relationships as well. We then use the typeORM API to pull the data from the database where we will then map it to a node that will be inserted into the graph database. To do this, we will use the OGM that is generated by GraphQL based off of the schemas we made. An OGM is an object-graph mapper that we use to map the data we pull from the MySQL database to the nodes that will be inserted into the graph database. For our graph database we will be using Neo4j, an enterprise-strength graph database (5). There are two major parts of a Neo4j graph database, nodes, and relationships.

In Neo4j graph databases nodes are entities that can have labels and properties. Labels are tags that can be applied to nodes to help represent the node's role in the graph domain. Node properties are very similar to fields in a schema in that they are key value pairs that store information about the node.

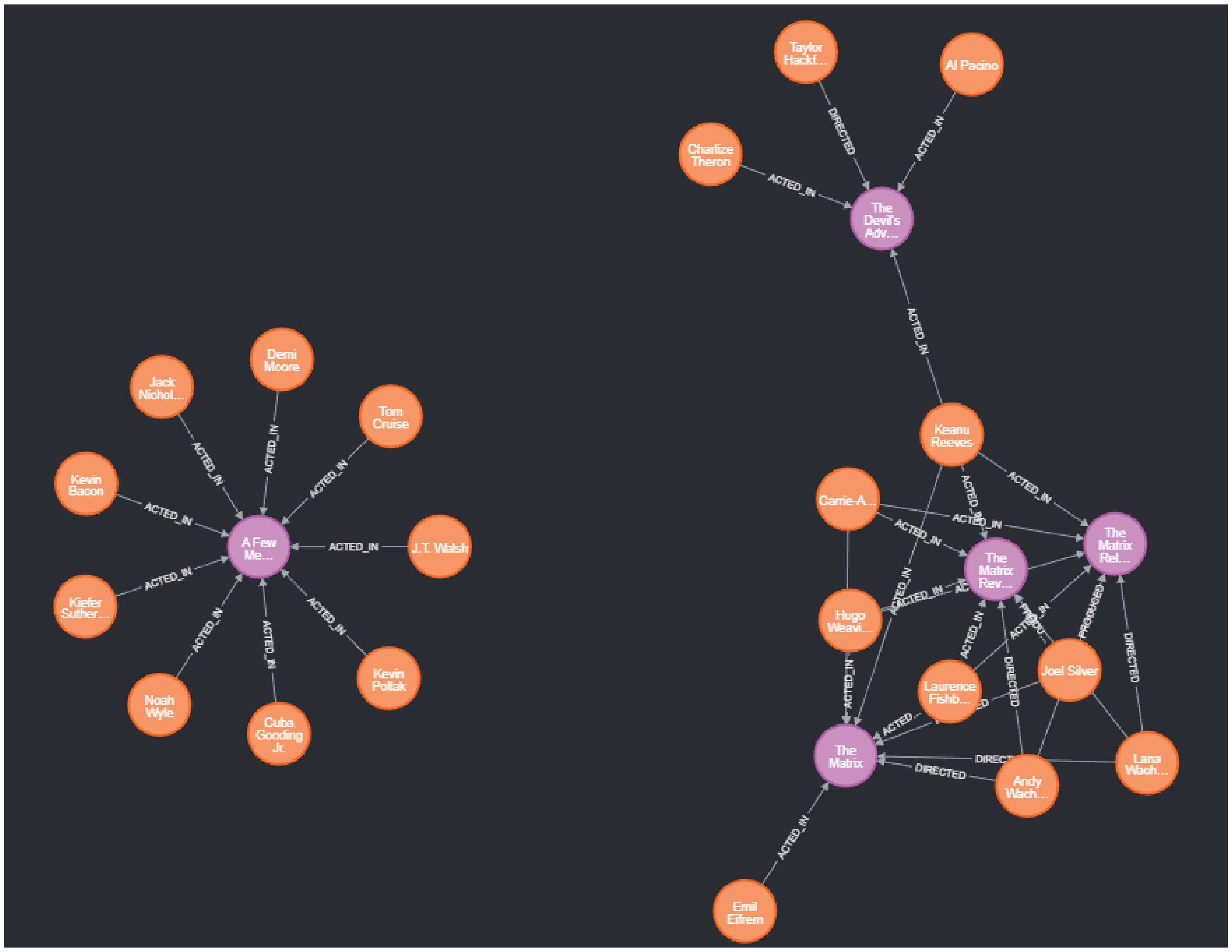
In Neo4j, relationships represent the edges of the graph. These relationships always have a start node, an end node, a direction, and a type. Like nodes, relationships can also have zero, one, or many properties.

If we consider the pet cat example from above, an intuitive graph can be gleaned from the schema. The Cat and Owner types would be the nodes, each holding properties that relate cleanly from the schema. The Cat node would have the properties name, age, and species. The Owner node would have the properties name and age. Ownership would be the relationship connecting the Cat and Owner nodes.

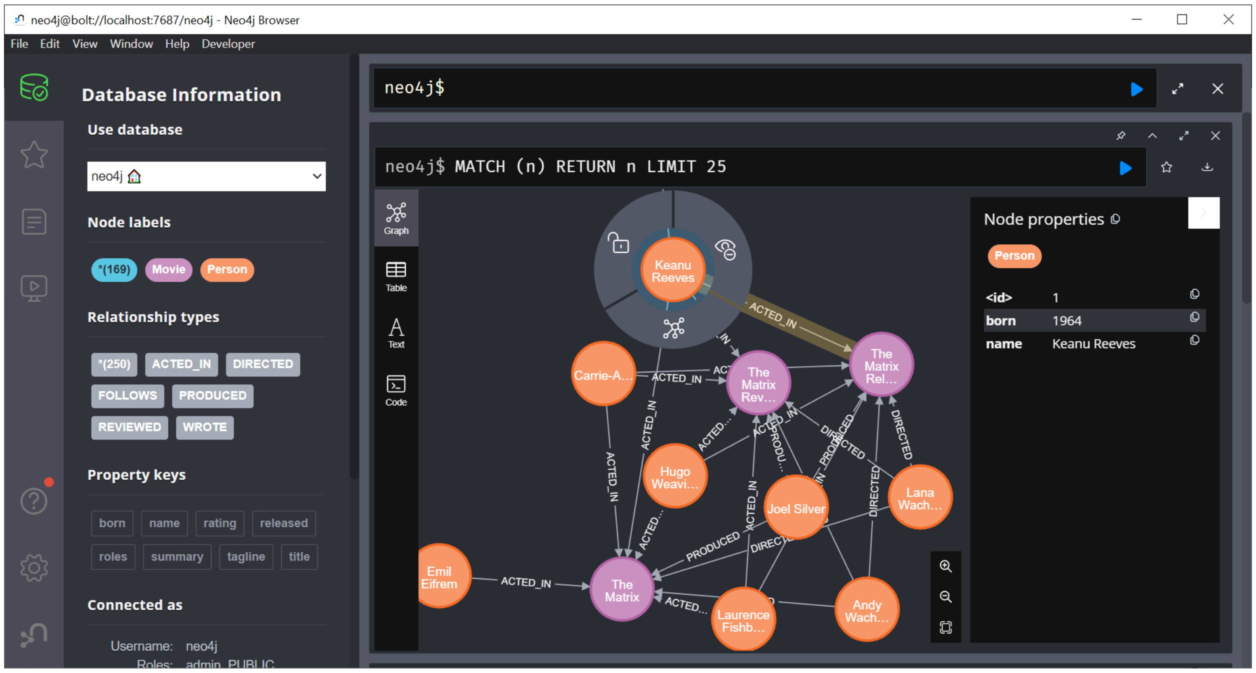
There are a few different options we could choose when implementing the relationship between cat and owner. First, it could point from Owner to Cat and be named OWNES. Second, it could be reversed, pointing from Cat to Owner, and be named OWNEDBY. Finally, because Neo4j's implementation allows nodes to have any number of relationships without sacrificing performance, we could choose to incorporate both previous implementations and have two relationships model ownership. Our small example graph would look like this:



Neo4j provides a more indepth graph database example. It models the relationships around movies.



Here is an example of an actor node and the fields it contains:



```
neo4j$
```

```
neo4j$ MATCH (n) RETURN n LIMIT 25
```

Use database: neo4j

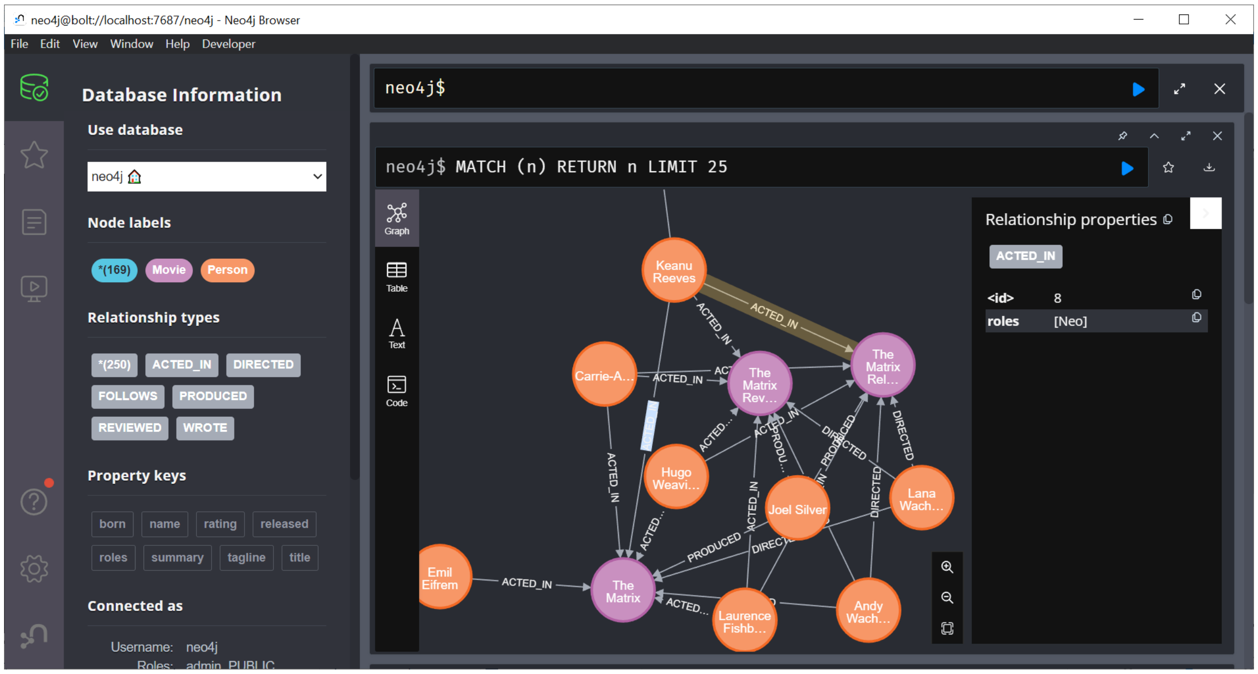
Username: neo4j
Roles: admin, PUBLIC

Node properties

Person

<id>	1
born	1964
name	Keanu Reeves

Followed by the ACTED_IN relationship with associated properties:



```
neo4j$
```

```
neo4j$ MATCH (n) RETURN n LIMIT 25
```

Use database: neo4j

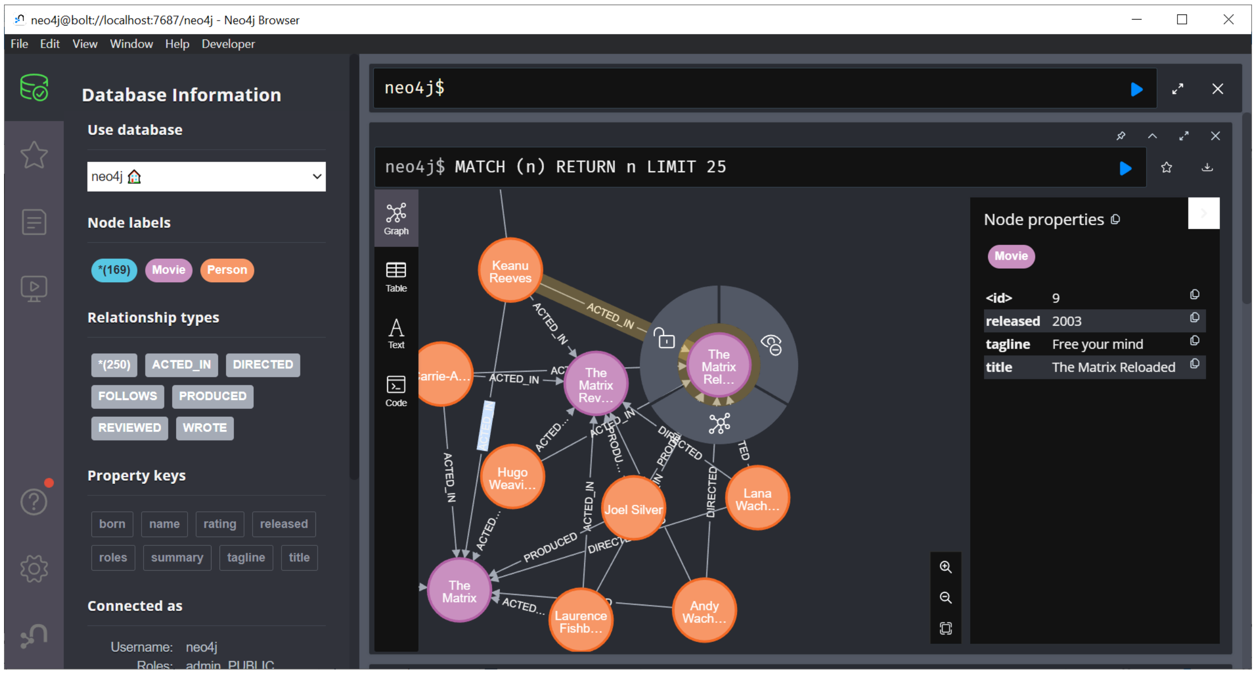
Username: neo4j
Roles: admin, PUBLIC

Relationship properties

ACTED_IN

<id>	8
roles	[Neo]

And finally, a Movie node with associated properties:



```
neo4j$ MATCH (n) RETURN n LIMIT 25
```

<id>	9
released	2003
tagline	Free your mind
title	The Matrix Reloaded

Although Neo4j has its own robust query language, our project will utilize GraphQL. GraphQL is a query language for both APIs and also a server side runtime. GraphQL will be utilized to complete all CRUD operations required for a graph database. GraphQL does this with two operations, Query and Mutate.

Query handles the Read operations of CRUD by allowing the user to write queries that return data in a JSON format. A query for our pet cat example would look like:

```
query GetCat($catName: "Baloo")  {
      Cat {
            name
            age
            species
            Owner {
                  name
            }
      }
}
```

Here, query lets GraphQL know the type of operation while GetCat specifies what type of query is being asked for. The query is using GetCat which asks for a name to identify which cat node we would like returned. Then in the Cat block we ask GraphQL to return the specific name, age, species, and owner fields for a cat with the name Baloo. Additionally, in the owner field we ask GraphQL to return information about the owner, their name. This very simple example shows the basics of GraphQL queries, but there are much more powerful tools to use as well. They can be found on the GraphQL website (3).

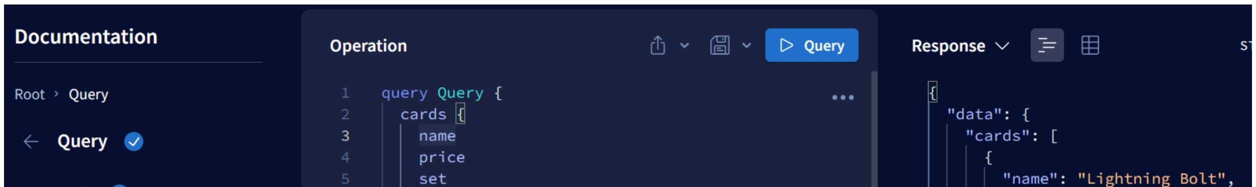
Mutations are similar to queries in execution. They have required fields, and return fields based on the user's request. However mutations are different in a very important way, they allow users to change data. This allows mutations to handle the Creat, Update, and Delete operations of CRUD. A Simple mutation for our pet cat example would look like:

```
mutation CreateCat($name: String!, $age: Int, $species: String,
$owner: Owner!) {
     createCat(name: $name, age: $age, species: $species, owner:
$owner) {
          name
          age
          species
          Owner
     }
}
```

Like the query call before, this call starts with mutation to denote what type of operation it will be. That is followed by the name of the mutation, which lists all the parameters passed in along with the types required for them. This is passed to createCat which uses the parameters to create a new cat. createCat also returns the new cat's name, age, species, and owner as defined in the mutation call. In addition to scalars like in the example above, you can also pass a mutation as an input object type. This is a special object that can be passed as an argument.

Another important difference between queries and mutations is how they execute. Queries are done in parallel, while mutations run in series to avoid possible race conditions.

Here is a concrete example of GraphQL connecting to a simple RestAPI with both a query and a mutation:



```
query Query {
  cards {
    name
    price
    set
```

```
{
  "data": {
    "cards": [
      {
        "name": "Lightning Bolt",
```

3. WORK SCHEDULE

Our work schedule began with the start of the spring semester. We did two week sprints to meet our milestone goals. Each two week sprint was split into week-long segments. The first week of each sprint had a planning day, and then coding and working on the milestone. The second week was dedicated to reviewing progress and fixing bugs. We met each morning of the week with our technical mentor, Max, to discuss all we did the day before and what our plan was for the next day.

Below is a timeline for our initial four milestones. Unfortunately, due to the fact that we were working on an internal system and it was the first time F5 has done a project like this with students, we weren't able to get access to the system until we received company laptops and were onboarded as interns. This finally happened on May 12th.

Sprint 1 - Schemas			
Section	**Task Name**	**Start Date**	**End Date**
Full Sprint	Create schemas	March 12th, 2023	March 26th, 2023
Planning/ Implementation	Write schema translations for JSON data to be used in data migration	March 12th, 2023	March 19th, 2023
Debug/Review	Review and critique JSON schema	March 19th, 2023	March 26th, 2023

The first sprint was focused on writing all the schema required for the data migration from the current relational database to the graph database. We split up the schema generation evenly between us three, with each person writing 10+ schemas and reviewing the ones written by the others.

Sprint 2 - Data Migration			
Section	**Task Name**	**Start Date**	**End Date**
Full Sprint	Implement Data Migration	March 26th, 2023	April 21st, 2023
Planning/ Implementation	Write program to extract JSON data from relational database and insert into graph database	March 26th, 2023	April 4th, 2023
Debug/Review	Review data migration program and fix bugs	April 4th, 2023	April 21st, 2023

The second sprint involved writing different functions that would preprocess the data we pulled out of the neo4j database, add a MD5 fingerprint hash to the data, as well as doing the same for any child nodes that these nodes had. We split up the writing of these functions evenly between the three of us. Due to not having access to the database until April 21st, we weren't able to test any of these functions until this past week.

Sprint 3/4 - Updating Backend/Frontend Logic			
Section	**Task Name**	**Start Date**	**End Date**
Full Sprint	Update TMS to point to the graph database	April 21st, 2023	May 21st, 2023
Planning/ Implementation	Update code in TMS to use new GraphQL service	April 21st, 2023	May 12th, 2023
Debug/Review	Review TMS changes and debug	May 12th, 2023	May 21st, 2023

As a result of losing a month due to onboarding, we have not been able to complete all of the goals that we have wanted to in the time that we have had. We plan on working until the end of May so that we can finish this project in its entirety and then present it to F5 at their tech day in May.

The third and fourth sprints are extended as the majority of the testing and debugging will take place in this sprint. In the third sprint, will be updating the logic for existing services to integrate with the new graph database. In the fourth sprint, we will focus on updating the backend and updating the front end, however some overlap is expected in these sprints.

Because the nature of diving into a new code base comes with required learning and familiarizing, we expect this sprint to require the most effort. As this is a relatively novel experience, when compared to college projects which are mostly ground up development, it should also prove to be excellent practice for real world development.

4. PROPOSAL STATEMENT

The crux of this project focuses on migration of both data and policies using GraphQL to make querying and mutating data easier, faster, and more reliable. Initially, we will be migrating JSON data from a relational database to a graphical Neo4j database using a migration script. Then, we will be modifying both the front end and back end of TMS to use a GraphQL API to manage data in both databases. This includes the migration of specified policies to function within the GraphQL API. Both of these steps will be split into four sprints.

The first step will be split into two separate sprints, the first covering writing schemas to be utilized by GraphQL to migrate JSON objects. The second sprint will be writing a migration script utilizing GraphQL to complete the migration of data from the TMS database to the Neo4j database.

The second step will also be split into two sprints, the first sprint will cover updating backend logic to first point TMS to the graphical database, then slowly migrate code in TMS to utilize GraphQL as an API gateway to both servers. The second sprint will update the frontend logic to now point to the GraphQL API allowing calls to be made directly to the GraphQL service, while some will still have to continue through to TMS.

These changes should improve the functionality of TMS in a few important ways. TMS's performance at runtime should be vastly improved because the relational database will no longer house duplicate data. Additionally, the JSON data that was stored in a relational database will now be stored in a more natural and efficient format in the Neo4j graph database. Furthermore, utilizing a GraphQL API allows for user friendly database management. Queries are written in a format very similar to JSON which means they are easily understandable for the developer.

4.1. Functional Requirements

Functional Requirement	Type of Source	Functional Requirement Description
1. Migration of Data and Code Base	JSON Data	● GraphQL schemas shall be defined for each JSON blob from the TMS connected database ● JSON blobs from the TMS connected database shall be parsed into the Neo4j database based on the defined schemas using a migration script
	Policies	● GraphQL schemas shall be defined for policies defining types, functionality, and data accessed as needed ● Policies shall be uploaded to the Neo4j databases as needed
2. Data Preservation	JSON Data	● When data is transferred, the data shall be confirmed correct through ability to compare the data from the Neo4j database to the TMS connected database to prevent data loss
3. Query	JSON Data	● The GraphQL API shall allow data to be queried from the Neo4j database
	Policies	● The GraphQL API shall allow specific policies to be queried from the Neo4j database ● When policies are queried they shall be distinguished by the logic performed and what data is used
4. Mutation	JSON Data	● The GraphQL API shall allow mutation of data from the Neo4j database including a recording of version history of such changes ● The GraphQL API mutation shall only affect the node of the graph that has been edited
	Policies	● The GraphQL API shall allow mutation of policies including a recording of version history of such

		changes ● When making mutations the GraphQL API shall only change the node of the policy being changed ● The GraphQL API shall allow policies to be directly uploaded to the Neo4j database
5. Source Code	Policies	● The GraphQL API shall allow policies to be directly downloaded as source code

4.2. Non-functional Requirements

Non-Functional Requirement	Functional Requirement Description
1. Caching	● How much memory to be cached and how often to clear the cache shall be defined ● Caching limits shall be defined as the point when the memory used starts to hinder the operating system
2. RAM	● A certain amount of RAM shall be required to satisfy caching requirements
3. CPU	● A minimum CPU of the system shall be required to reasonably handle service computation in a timely manner
4. Storage	● A minimum amount of disk space shall be required to satisfy the GraphQL API and Neo4j database ● If a larger memory capacity is required than is common on most systems (8 - 16 GBs) a solid state drive shall be required to satisfy caching requirements

4.3. Performance Requirements

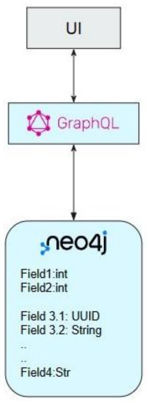
The GraphQL API, TMS API, and the Neo4j database are intended to be running at all times, with specified downtime for upgrades and maintenance if needed.

Most developers will be using the Mac operating system, but others may use Windows. These differences shall not affect the performance of the services. The GraphQL API, in both cases, must supply near immediate feedback for data and policy queries and must nearly immediately and sequentially supply mutations for approval from an administrator.

Limiting the necessary memory requirements for running services locally will be a main factor in the building process. Fast response times are necessary to prevent a hindrance on the developer waiting for processes to complete, so a balance between hardware requirements and its effect on performance will be measured using the Neo4j database along with the GraphQL API.

4.4. Interface requirements

This project is based on interface implementation as we are evolving the interface architecture of the current TMS system. This interface is going to have to not only find and return specific data from the respective database, but also respond to commands for adding, updating, and deleting data from the database. The GQL Gateway API plays a huge part in this system and the ability for it to execute requests as it is the part that decides where to find the data and possibly decide which database to call on if more databases are added to the system.



4.5. Development standards

This project will use Test-driven development supplemented with static linting and code reviews as development standards.

The test-driven development cycle has five major steps:

1. Add a new test for a new or existing feature
    - 1.1. This new test should pass if the feature's requirements are met. This is advantageous as it makes us focus on the requirements of a feature before we even begin coding.
2. Run all the tests that have been written, including the new test.
    - 2.1. The new test should fail, which demonstrates the need for new code to be written. Although this might seem like a waste of a run, it is also a good check to make sure the test framework is working as intended.
3. Write simple code to pass the new test.
    - 3.1. Messy code, and even hard coded solutions are acceptable in this step.
4. Run all the tests again checking that they all pass.
    - 4.1. If any tests fail at this step code will be revised until all the tests pass. This step is done to make sure new code does not break any previously working features.
5. Refactor the code where required, testing after each new change.
    - 5.1. If hard coded data still exists in this step it should be removed or replaced. As in step 4 testing after each change is important to check that no existing functionality has been broken.

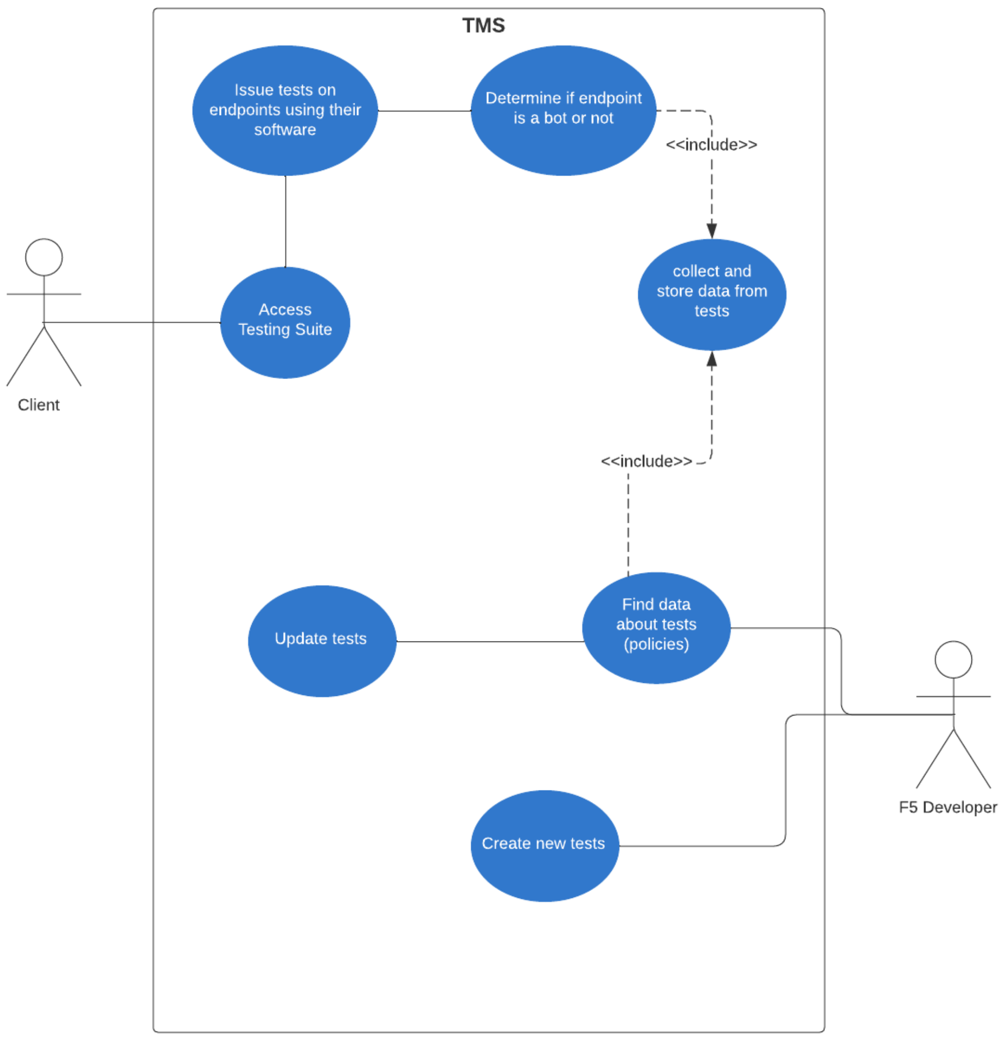
This cycle then repeats any time new functionality is added. To keep debugging to a minimum, tests should be small and written often. This way if something breaks it will be much easier to identify at what point the code was changed to cause the failure.

To supplement this development style we will be using standard coding practices like standard naming conventions and linting. Because our project will be written in TypeScript we will be following the rules provided by F5 using ESLint. Naming conventions will also follow F5's desired guidelines.

4.6. Architectural Design Documents

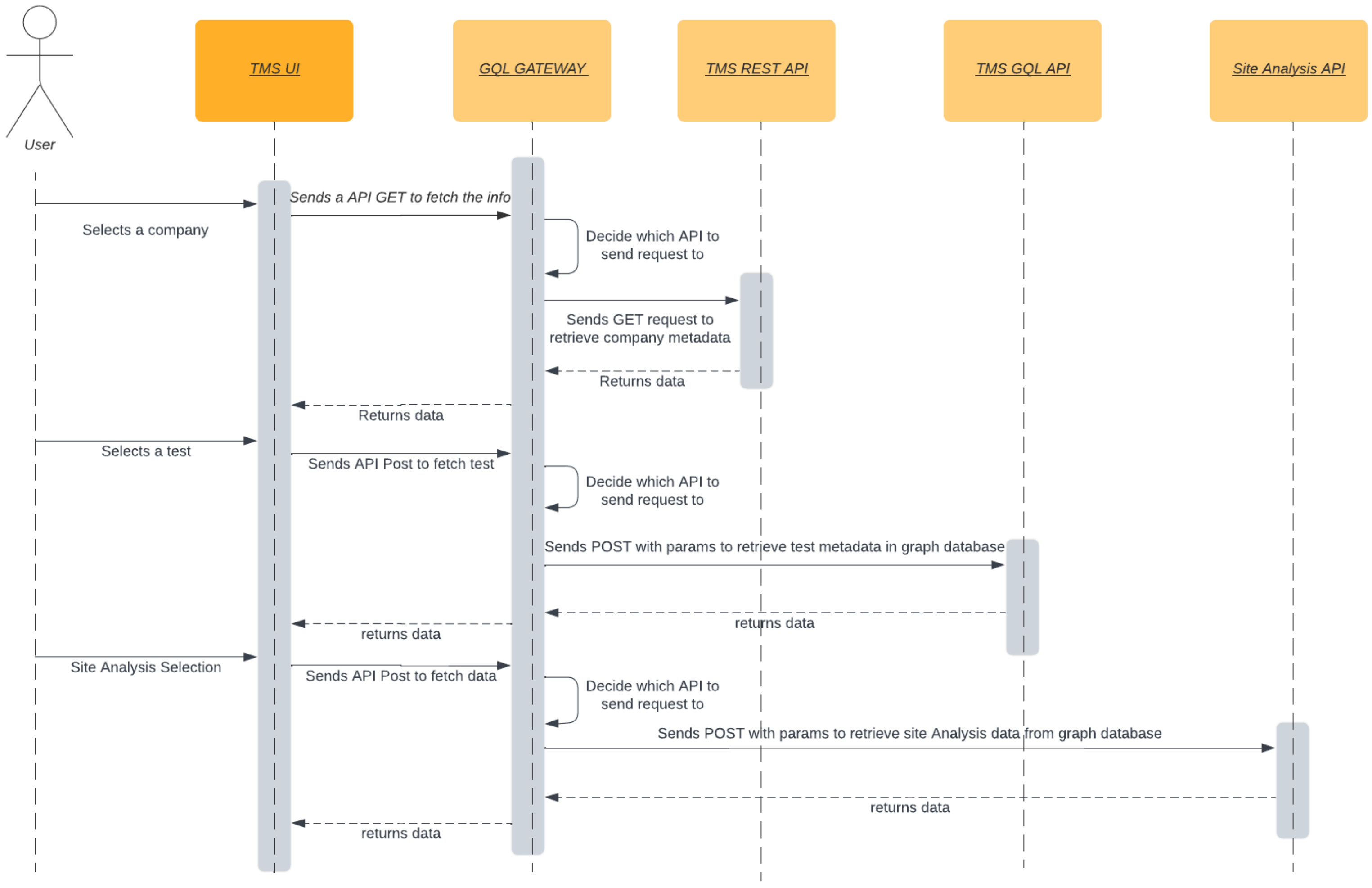
4.6.1 Use Case Diagram

The simple Use Case Diagram below represents a very high level view of the TMS system but more importantly represents how data is collected and then used by F5 developers to improve these tests. It also shows the importance to improve access to this collected data which is done by the use of a graph database and GraphQL API.


TMS
Issue tests on endpoints using their software
Determine if endpoint is a bot or not
<<include>>
collect and store data from tests
Access Testing Suite
Client
<<include>>
Update tests
Find data about tests (policies)
F5 Developer
Create new tests

4.6.2 Sequence Diagram

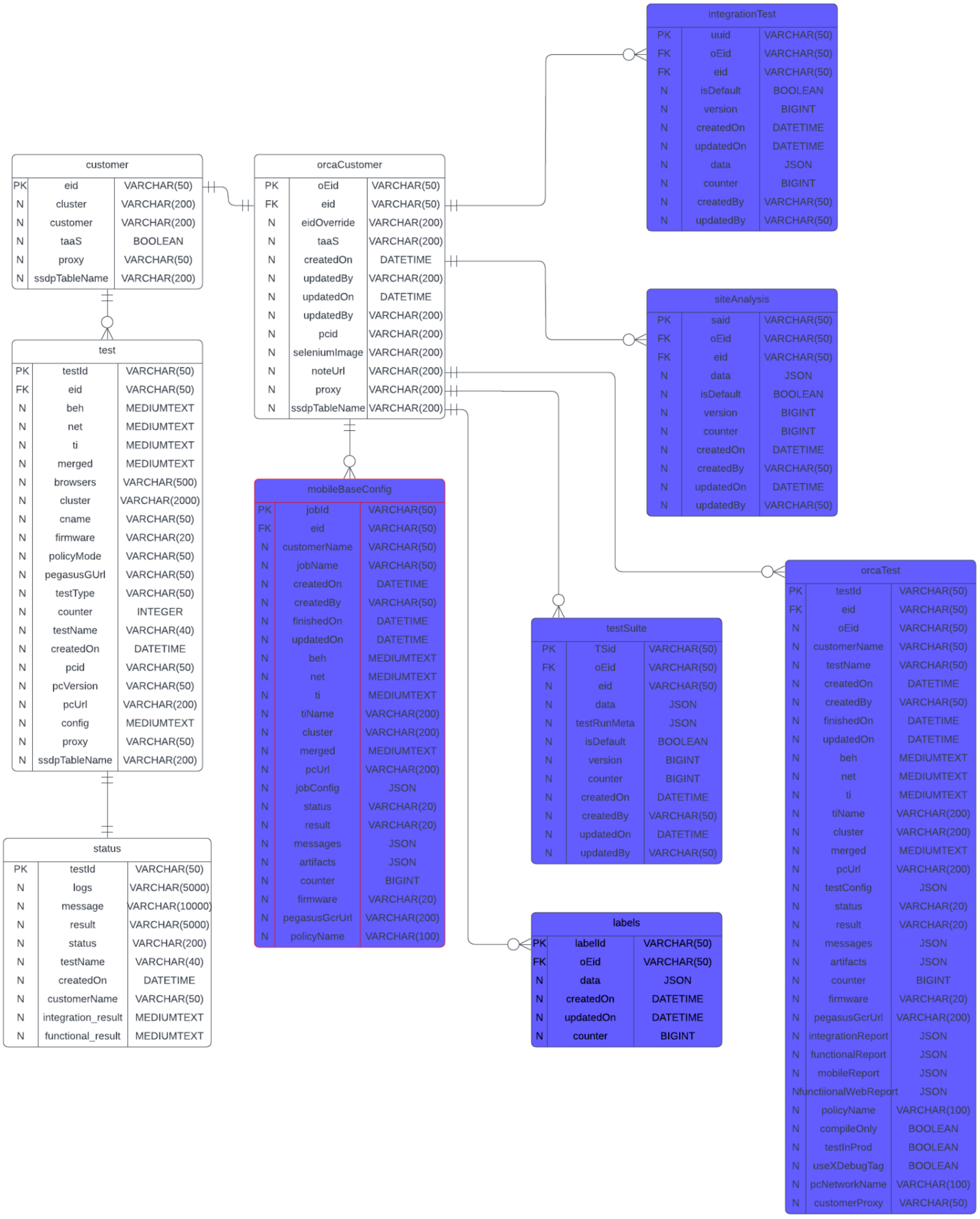
The sequence diagram below shows the function of the GQL Gateway and the interaction between it and the various APIs used to retrieve data from the databases. The GQL gateway receives an HTML request and then decides which API to send the request to.



User
TMS UI
GQL GATEWAY
TMS REST API
TMS GQL API
Site Analysis API
Selects a company
Sends a API GET to fetch the info
Decide which API to send request to
Sends GET request to retrieve company metadata
Returns data
Returns data
Selects a test
Sends API Post to fetch test
Decide which API to send request to
Sends POST with params to retrieve test metadata in graph database
returns data
returns data
Site Analysis Selection
Sends API Post to fetch data
Decide which API to send request to
Sends POST with params to retrieve site Analysis data from graph database
returns data
returns data

4.6.3 Old ERD

This first ER (Entity Relationship) diagram below is the ER diagram for the current tabular database used in the TMS stack. The entities in this database that we plan on changing and migrating the JSON data out of are highlighted in blue. Note the fields where the type is JSON. Those are going to be replaced with Ids that we will use in a GraphQL query to return the JSON objects that were stored in those positions.



customer

PK	eid	VARCHAR(50)
N	cluster	VARCHAR(200)
N	customer	VARCHAR(200)
N	taaS	BOOLEAN
N	proxy	VARCHAR(50)
N	ssdpTableName	VARCHAR(200)

test

PK	testId	VARCHAR(50)
FK	eid	VARCHAR(50)
N	beh	MEDIUMTEXT
N	net	MEDIUMTEXT
N	ti	MEDIUMTEXT
N	merged	MEDIUMTEXT
N	browsers	VARCHAR(500)
N	cluster	VARCHAR(2000)
N	cname	VARCHAR(50)
N	firmware	VARCHAR(20)
N	policyMode	VARCHAR(50)
N	pegasusGUrl	VARCHAR(50)
N	testType	VARCHAR(50)
N	counter	INTEGER
N	testName	VARCHAR(40)
N	createdOn	DATETIME
N	pcid	VARCHAR(50)
N	pcVersion	VARCHAR(50)
N	pcUrl	VARCHAR(200)
N	config	MEDIUMTEXT
N	proxy	VARCHAR(50)
N	ssdpTableName	VARCHAR(200)

status

PK	testId	VARCHAR(50)
N	logs	VARCHAR(5000)
N	message	VARCHAR(10000)
N	result	VARCHAR(5000)
N	status	VARCHAR(200)
N	testName	VARCHAR(40)
N	createdOn	DATETIME
N	customerName	VARCHAR(50)
N	integration_result	MEDIUMTEXT
N	functional_result	MEDIUMTEXT

orcaCustomer

PK	oEid	VARCHAR(50)
FK	eid	VARCHAR(50)
N	eidOverride	VARCHAR(200)
N	taaS	VARCHAR(200)
N	createdOn	DATETIME
N	updatedBy	VARCHAR(200)
N	updatedOn	DATETIME
N	updatedBy	VARCHAR(200)
N	pcid	VARCHAR(200)
N	seleniumImage	VARCHAR(200)
N	noteUrl	VARCHAR(200)
N	proxy	VARCHAR(200)
N	ssdpTableName	VARCHAR(200)

mobileBaseConfig

PK	jobId	VARCHAR(50)
FK	eid	VARCHAR(50)
N	customerName	VARCHAR(50)
N	jobName	VARCHAR(50)
N	createdOn	DATETIME
N	createdBy	VARCHAR(50)
N	finishedOn	DATETIME
N	updatedOn	DATETIME
N	beh	MEDIUMTEXT
N	net	MEDIUMTEXT
N	ti	MEDIUMTEXT
N	tiName	VARCHAR(200)
N	cluster	VARCHAR(200)
N	merged	MEDIUMTEXT
N	pcUrl	VARCHAR(200)
N	jobConfig	JSON
N	status	VARCHAR(20)
N	result	VARCHAR(20)
N	messages	JSON
N	artifacts	JSON
N	counter	BIGINT
N	firmware	VARCHAR(20)
N	pegasusGcrUrl	VARCHAR(200)
N	policyName	VARCHAR(100)

integrationTest

PK	uuid	VARCHAR(50)
FK	oEid	VARCHAR(50)
FK	eid	VARCHAR(50)
N	isDefault	BOOLEAN
N	version	BIGINT
N	createdOn	DATETIME
N	updatedOn	DATETIME
N	data	JSON
N	counter	BIGINT
N	createdBy	VARCHAR(50)
N	updatedBy	VARCHAR(50)

siteAnalysis

PK	said	VARCHAR(50)
FK	oEid	VARCHAR(50)
FK	eid	VARCHAR(50)
N	data	JSON
N	isDefault	BOOLEAN
N	version	BIGINT
N	counter	BIGINT
N	createdOn	DATETIME
N	createdBy	VARCHAR(50)
N	updatedOn	DATETIME
N	updatedBy	VARCHAR(50)

testSuite

PK	TSid	VARCHAR(50)
FK	oEid	VARCHAR(50)
N	eid	VARCHAR(50)
N	data	JSON
N	testRunMeta	JSON
N	isDefault	BOOLEAN
N	version	BIGINT
N	counter	BIGINT
N	createdOn	DATETIME
N	createdBy	VARCHAR(50)
N	updatedOn	DATETIME
N	updatedBy	VARCHAR(50)

labels

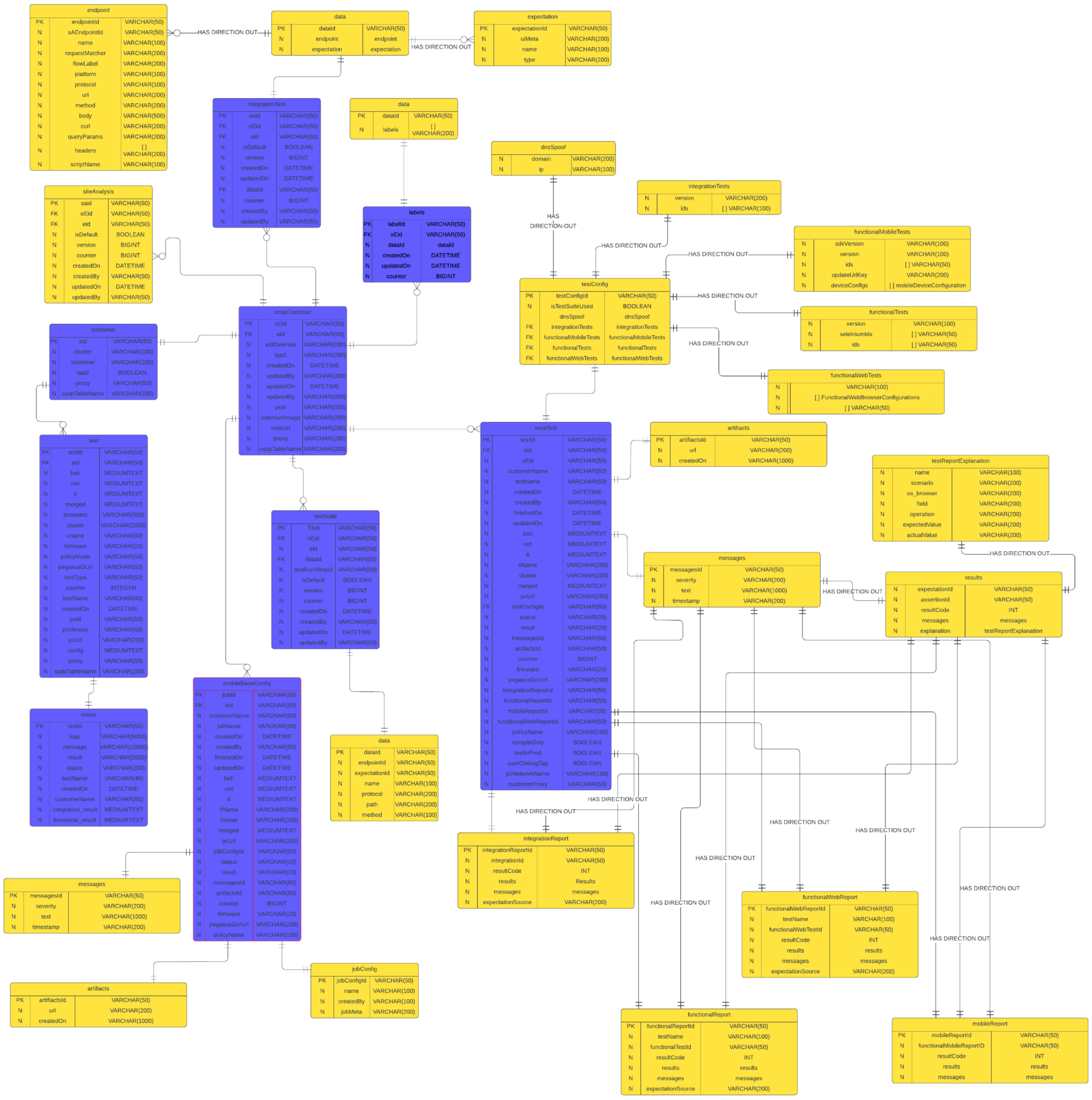
PK	labelId	VARCHAR(50)
FK	oEid	VARCHAR(50)
N	data	JSON
N	createdOn	DATETIME
N	updatedOn	DATETIME
N	counter	BIGINT

orcaTest

PK	testId	VARCHAR(50)
FK	eid	VARCHAR(50)
N	oEid	VARCHAR(50)
N	customerName	VARCHAR(50)
N	testName	VARCHAR(50)
N	createdOn	DATETIME
N	createdBy	VARCHAR(50)
N	finishedOn	DATETIME
N	updatedOn	DATETIME
N	beh	MEDIUMTEXT
N	net	MEDIUMTEXT
N	ti	MEDIUMTEXT
N	tiName	VARCHAR(200)
N	cluster	VARCHAR(200)
N	merged	MEDIUMTEXT
N	pcUrl	VARCHAR(200)
N	testConfig	JSON
N	status	VARCHAR(20)
N	result	VARCHAR(20)
N	messages	JSON
N	artifacts	JSON
N	counter	BIGINT
N	firmware	VARCHAR(20)
N	pegasusGcrUrl	VARCHAR(200)
N	integrationReport	JSON
N	functionalReport	JSON
N	mobileReport	JSON
N	functiionalWebReport	JSON
N	policyName	VARCHAR(100)
N	compileOnly	BOOLEAN
N	testInProd	BOOLEAN
N	useXDebugTag	BOOLEAN
N	pcNetworkName	VARCHAR(100)
N	customerProxy	VARCHAR(50)

4.6.4 New ERD

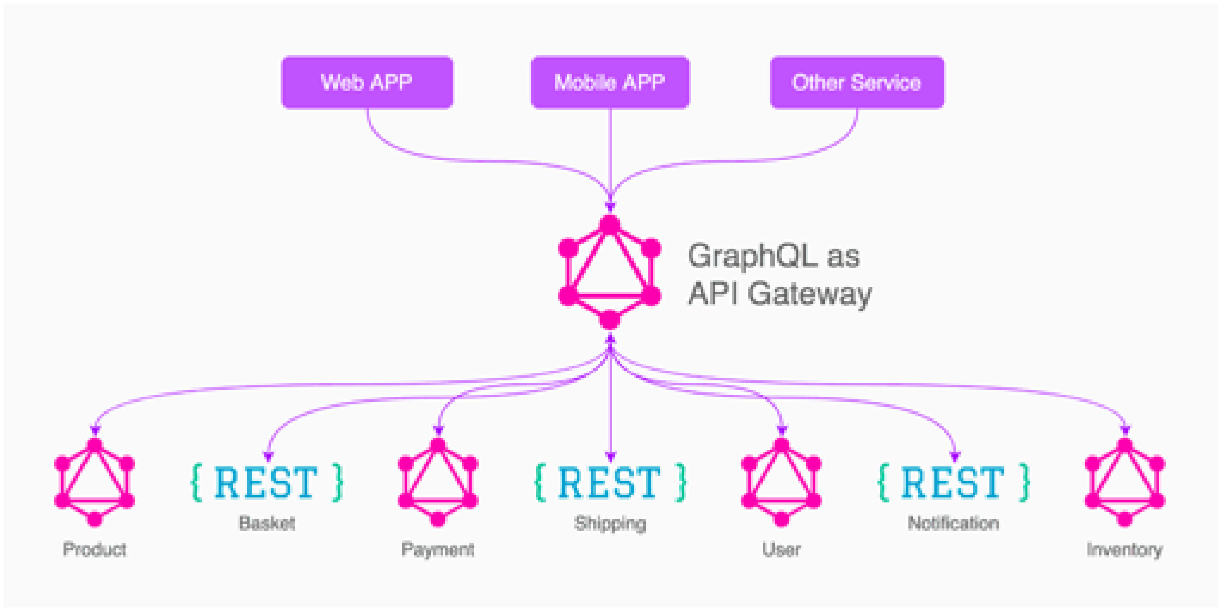
On the new ERD below, the blue entities still represent the data that is contained in the tabular database, however, the new yellow entities are going to be what is contained in the new graph database. Note that ids are used in place of the JSON data in the blue entities so that we have a gateway to enter the graph data by using these ids in GQL queries. Also note that the entirety of the siteAnalysis entity is being migrated into a graph database as this is being transformed into a separate microservice queried by the GQL gateway API. The graph database is directional so directions have been added to the relationships in which there are directions.   All of these directed relationships are HAS and directed toward the entity that "HAS" an instance of the other entity.



endpoint
PK endpointId VARCHAR(50)
N sAEndpointId VARCHAR(50)
N name VARCHAR(100)
N requestMatcher VARCHAR(200)
N flowLabel VARCHAR(200)
N platform VARCHAR(100)
N protocol VARCHAR(100)
N url VARCHAR(200)
N method VARCHAR(200)
N body VARCHAR(500)
N curl VARCHAR(200)
N queryParams VARCHAR(200)
N headers [ ] VARCHAR(200)
N scriptName VARCHAR(100)
HAS DIRECTION OUT
data
PK dataId VARCHAR(50)
N endpoint endpoint
N expectation expectation
HAS DIRECTION OUT
expectation
PK expectationId VARCHAR(50)
N uiMeta VARCHAR(200)
N name VARCHAR(100)
N type VARCHAR(200)
integrationTest
PK uuid VARCHAR(50)
FK oEid VARCHAR(50)
FK eid VARCHAR(50)
N isDefault BOOLEAN
N version BIGINT
N createdOn DATETIME
N updatedOn DATETIME
FK dataId VARCHAR(50)
N counter BIGINT
N createdBy VARCHAR(50)
N updatedBy VARCHAR(50)
data
PK dataId VARCHAR(50)
N labels [ ] VARCHAR(200)
dnsSpoof
N domain VARCHAR(200)
N ip VARCHAR(100)
HAS DIRECTION-OUT
integrationTests
N version VARCHAR(200)
N ids [ ] VARCHAR(100)
siteAnalysis
PK saId VARCHAR(50)
FK oEid VARCHAR(50)
FK eid VARCHAR(50)
N isDefault BOOLEAN
N version BIGINT
N counter BIGINT
N createdOn DATETIME
N createdBy VARCHAR(50)
N updatedOn DATETIME
N updatedBy VARCHAR(50)
labels
PK labelId VARCHAR(50)
FK oEid VARCHAR(50)
N dataId dataId
N createdOn DATETIME
N updatedOn DATETIME
N counter BIGINT
HAS DIRECTION OUT
HAS DIRECTION OUT
functionalMobileTests
N sdkVersion VARCHAR(100)
N version VARCHAR(100)
N ids [ ] VARCHAR(50)
N updateUrlKey VARCHAR(200)
N deviceConfigs [ ] mobileDeviceConfiguration
testConfig
PK testConfigId VARCHAR(50)
N isTestSuiteUsed BOOLEAN
dnsSpoof dnsSpoof
FK integrationTests integrationTests
FK functionalMobileTests functionalMobileTests
FK functionalTests functionalTests
FK functionalWebTests functionalWebTests
HAS DIRECTION OUT
functionalTests
N version VARCHAR(100)
N seleniumIds [ ] VARCHAR(50)
N ids [ ] VARCHAR(50)
HAS DIRECTION OUT
functionalWebTests
N VARCHAR(100)
N [ ] FunctionalWebBrowserConfigurations
N [ ] VARCHAR(50)
customer
PK eid VARCHAR(50)
N cluster VARCHAR(200)
N customer VARCHAR(200)
N taaS BOOLEAN
N proxy VARCHAR(50)
N ssdpTableName VARCHAR(200)
orcaCustomer
PK oEid VARCHAR(50)
FK eid VARCHAR(50)
N eidOverride VARCHAR(200)
N taaS VARCHAR(200)
N createdOn DATETIME
N updatedBy VARCHAR(200)
N updatedOn DATETIME
N updatedBy VARCHAR(200)
N pcid VARCHAR(200)
N seleniumImage VARCHAR(200)
N noteUrl VARCHAR(200)
N proxy VARCHAR(200)
N ssdpTableName VARCHAR(200)
orcaTest
PK testId VARCHAR(50)
FK eid VARCHAR(50)
N oEid VARCHAR(50)
N customerName VARCHAR(50)
N testName VARCHAR(50)
N createdOn DATETIME
N createdBy VARCHAR(50)
N finishedOn DATETIME
N updatedOn DATETIME
N beh MEDIUMTEXT
N net MEDIUMTEXT
N ti MEDIUMTEXT
N tiName VARCHAR(200)
N cluster VARCHAR(200)
N merged MEDIUMTEXT
N pcUrl VARCHAR(200)
FK testConfigId VARCHAR(50)
N status VARCHAR(20)
N result VARCHAR(20)
N messagesId VARCHAR(50)
N artifactsId VARCHAR(50)
N counter BIGINT
N firmware VARCHAR(20)
N pegasusGcrUrl VARCHAR(200)
N integrationReportId VARCHAR(50)
N functionalReportId VARCHAR(50)
N mobileReportId VARCHAT(50)
N functionalWebReportId VARCHAR(50)
N policyName VARCHAR(100)
N compileOnly BOOLEAN
N testInProd BOOLEAN
N useXDebugTag BOOLEAN
N pcNetworkName VARCHAR(100)
N customerProxy VARCHAR(50)
artifacts
PK artifactsId VARCHAR(50)
N url VARCHAR(200)
N createdOn VARCHAR(1000)
testReportExplanation
N name VARCHAR(100)
N scenario VARCHAR(200)
N os_browser VARCHAR(200)
N field VARCHAR(200)
N operation VARCHAR(200)
N expectedValue VARCHAR(200)
N actualValue VARCHAR(200)
HAS DIRECTION OUT
test
PK testId VARCHAR(50)
FK eid VARCHAR(50)
N beh MEDIUMTEXT
N net MEDIUMTEXT
N ti MEDIUMTEXT
N merged MEDIUMTEXT
N browsers VARCHAR(500)
N cluster VARCHAR(2000)
N cname VARCHAR(50)
N firmware VARCHAR(20)
N policyMode VARCHAR(50)
N pegasusGUrl VARCHAR(50)
N testType VARCHAR(50)
N counter INTEGER
N testName VARCHAR(40)
N createdOn DATETIME
N pcid VARCHAR(50)
N pcVersion VARCHAR(50)
N pcUrl VARCHAR(200)
N config MEDIUMTEXT
N proxy VARCHAR(50)
N ssdpTableName VARCHAR(200)
testSuite
PK TSid VARCHAR(50)
FK oEid VARCHAR(50)
N eid VARCHAR(50)
FK dataId VARCHAR(50)
N testRunMetaId VARCHAR(50)
N isDefault BOOLEAN
N version BIGINT
N counter BIGINT
N createdOn DATETIME
N createdBy VARCHAR(50)
N updatedOn DATETIME
N updatedBy VARCHAR(50)
messages
PK messagesId VARCHAR(50)
N severity VARCHAR(200)
N text VARCHAR(1000)
N timestamp VARCHAR(200)
HAS DIRECTION OUT
results
N expectationId VARCHAR(50)
N assertionId VARCHAR(50)
N resultCode INT
N messages messages
N explanation testReportExplanation
status
PK testId VARCHAR(50)
N logs VARCHAR(5000)
N message VARCHAR(10000)
N result VARCHAR(5000)
N status VARCHAR(200)
N testName VARCHAR(40)
N createdOn DATETIME
N customerName VARCHAR(50)
N integration_result MEDIUMTEXT
N functional_result MEDIUMTEXT
mobileBaseConfig
PK jobId VARCHAR(50)
FK eid VARCHAR(50)
N customerName VARCHAR(50)
N jobName VARCHAR(50)
N createdOn DATETIME
N createdBy VARCHAR(50)
N finishedOn DATETIME
N updatedOn DATETIME
N beh MEDIUMTEXT
N net MEDIUMTEXT
N ti MEDIUMTEXT
N tiName VARCHAR(200)
N cluster VARCHAR(200)
N merged MEDIUMTEXT
N pcUrl VARCHAR(200)
N jobConfigId VARCHAR(50)
N status VARCHAR(20)
N result VARCHAR(20)
N messagesId VARCHAR(50)
N artifactsId VARCHAR(50)
N counter BIGINT
N firmware VARCHAR(20)
N pegasusGcrUrl VARCHAR(200)
N policyName VARCHAR(100)
data
PK dataId VARCHAR(50)
N endpointId VARCHAR(50)
N expectationId VARCHAR(50)
N name VARCHAR(100)
N protocol VARCHAR(200)
N path VARCHAR(200)
N method VARCHAR(100)
HAS DIRECTION OUT
HAS DIRECTION OUT
integrationReport
PK integrationReportId VARCHAR(50)
N integrationId VARCHAR(50)
N resultCode INT
N results Results
N messages messages
N expectationSource VARCHAR(200)
HAS DIRECTION OUT
HAS DIRECTION OUT
HAS DIRECTION OUT
functionalWebReport
PK functionalWebReportId VARCHAR(50)
N testName VARCHAR(100)
N functionalWebTestId VARCHAR(50)
N resultCode INT
N results results
N messages messages
N expectationSource VARCHAR(200)
HAS DIRECTION OUT
HAS DIRECTION OUT
HAS DIRECTION OUT
messages
PK messagesId VARCHAR(50)
N severity VARCHAR(200)
N text VARCHAR(1000)
N timestamp VARCHAR(200)
jobConfig
PK jobConfigId VARCHAR(50)
N name VARCHAR(100)
N createdBy VARCHAR(100)
N jobMeta VARCHAR(200)
artifacts
PK artifactsId VARCHAR(50)
N url VARCHAR(200)
N createdOn VARCHAR(1000)
functionalReport
PK functionalReportId VARCHAR(50)
N testName VARCHAR(100)
N functionalTestId VARCHAR(50)
N resultCode INT
N results results
N messages messages
N expectationSource VARCHAR(200)
mobileReport
PK mobileReportId VARCHAR(50)
N functionalMobileReportID VARCHAR(50)
N resultCode INT
N results results
N messages messages

4.6.5 API Gateway Pattern

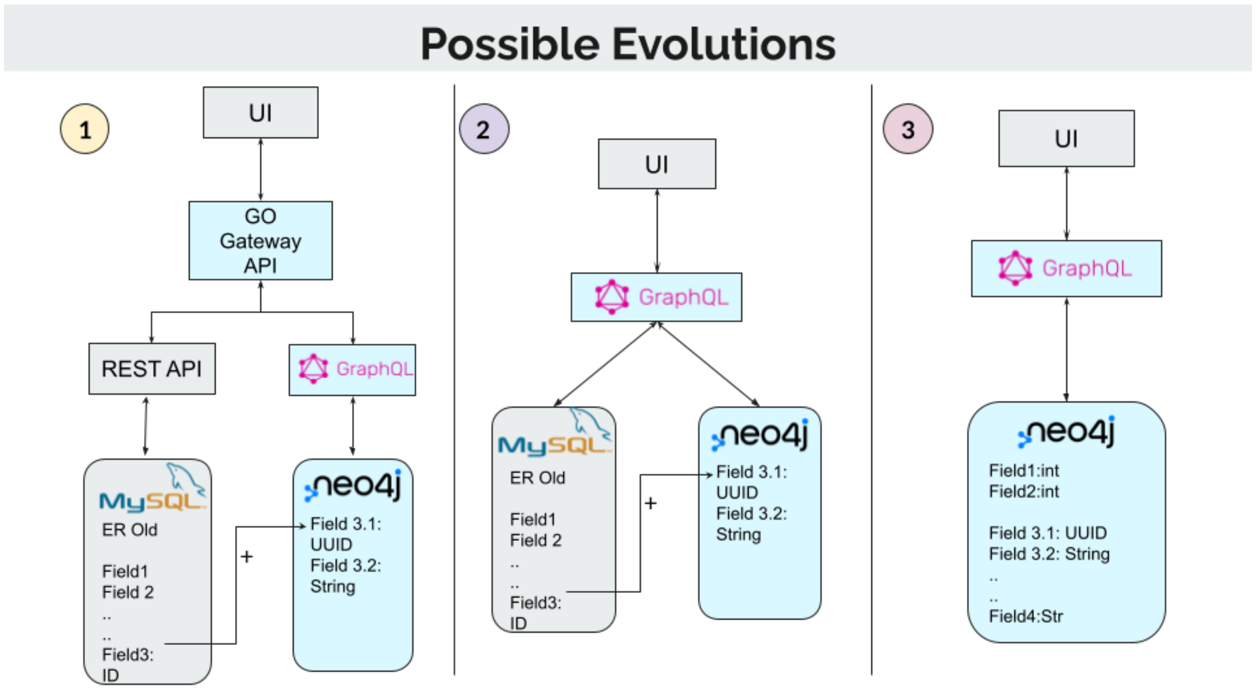
This system will implement the API gateway pattern using a GraphQL API as the gateway to the Neo4j database. The advantage of having a GraphQL API is that it has the ability, as seen in the sequence diagram on page 21, to decide where to send the request from the Client. We have decided to move all data to a single database, but this GraphQL API allows for more databases or APIs to be added and queried in a single query, which speeds up the runtime compared to REST.



(6)

4.7. Design Tradeoffs

Before progress was to be made, the choice arose on how to structure and implement the new database along with a GraphQL API. There were three possible evolutions of the system, each with their benefits and drawbacks. The first option was to modify the current API to interact with either the original or GraphQL API, where the GraphQL API alone would handle connection to the Neo4j database. The second option was to only migrate certain portions of data to the Neo4j database and have a GraphQL API that would connect to either the old or new database depending on where the data lives. The third option was to have a complete transition to only utilize a GraphQL API and Neo4j database, including migration of all data over to the Neo4j database.



The first option would be the least amount of work necessary, but much of the policies and data would not benefit from the improvements that GraphQL and Neo4j bring. The second option would bring about all the features from GraphQL and Neo4j, but more logic would have to be added in order for the GraphQL API to know which database the required data lives in. The third option, the one that was decided upon, brought about the most work, but made the most sense to implement. It was decided to be most beneficial to have all data migrated over to Neo4j since we were doing any data migration in the first place, thus it made the most sense to also transfer all API functionality to be handled by the GraphQL API. This will allow future functionality of the system to access all features and benefits of a GraphQL API paired with a Neo4j database, but will be at the cost of being more prone to mistakes or errors that would break functionality, increased workload of migration and API integration, and increased workload of policy transitions.

Another decision that brought about benefits and pitfalls was how to implement the preprocessing functions. These functions are designed to traverse the database and create the tree-like structure to be used in the Neo4j database for each type. A separate function was needed for each designated type in the Schema in order to create the necessary structure, which would have amounted to 57 functions to be hand coded. This would have been an easy yet tedious task, as each function follows a very similar format by which a fingerprint is created and the structure is formed, but any time changes were made to the original schema, changes to these functions were required as well. Instead, it was decided that we could utilize the typescript factory method to automatically generate these functions, diverting the tedious repetition of function creation to the utilization of typescript. This was a drawback as further research and implementation was required, but overall a benefit as the tedious nature of this section was avoided along with being possible to run the function multiple times any time changes are required to the original schema.

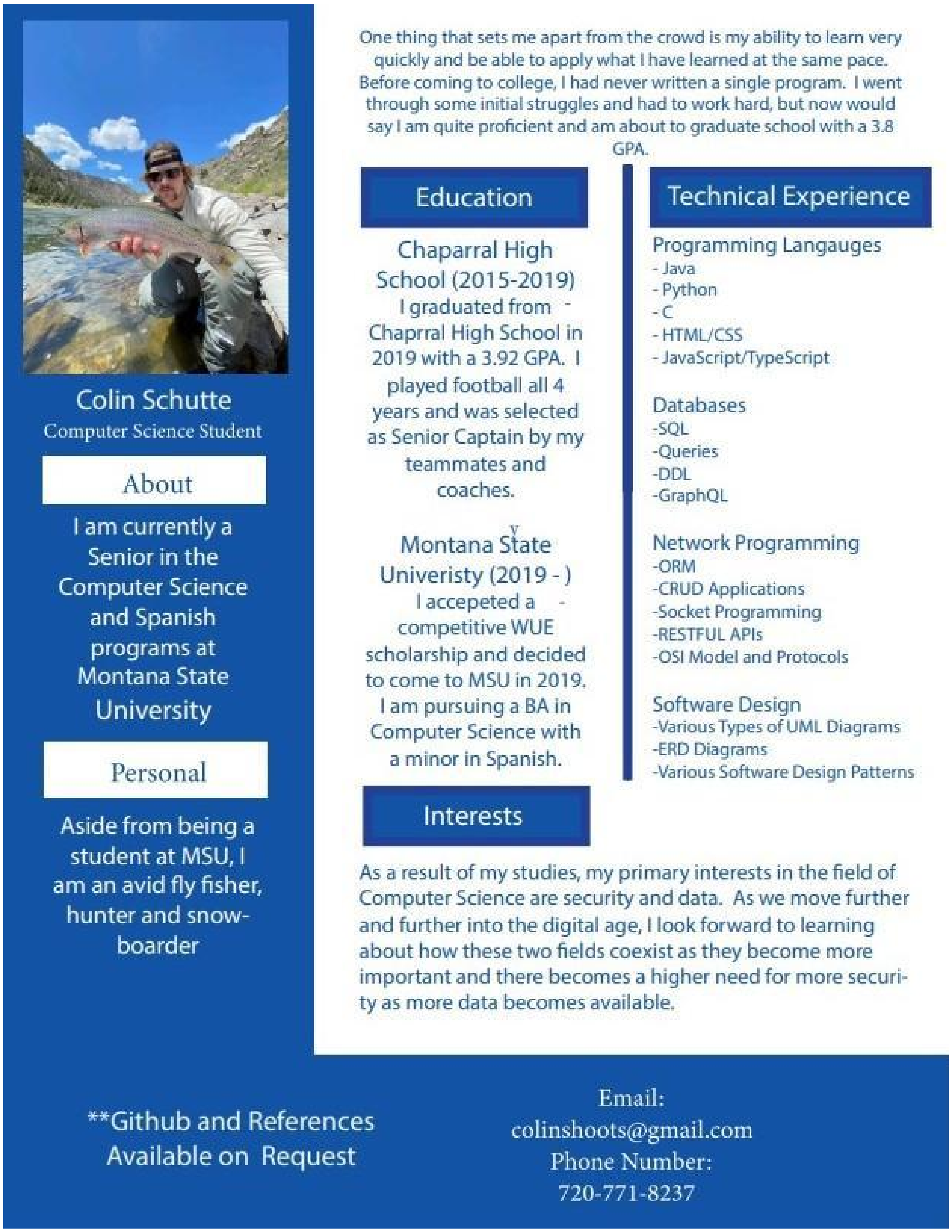
5. RESULTS

As we are still planning on working on this project for another month, we expect the completion of the project to result in improvements for both users and developers.

By the end of the project we expect to have migrated data from the current relational database to a new graph database with GraphQL utilizing the schemas we will be writing. As a result, 22 new “entities” will be exposed which are actually just nodes in the neo4j database. This can be seen in the old and new ER diagrams on pages 22 and 24. A majority of the data in the database will now be exposed and no longer hidden as JSON blobs. In addition, we will have updated the frontend and backend logic of the current project to use a GraphQL API as a gateway to the services.
Due to the efficiency improvements of navigating the JSON data to a graph database, runtime will be improved. Improved runtime will translate to a better user experience, as responsiveness on the front end will be improved as well. Also the user will benefit from better turnaround time from developers due to the user friendly nature of the new technology.

Developers should see improvements in database management, usability, database scaling, and performance. Developers will benefit greatly from the ease of use of GraphQL to manage both the new graph database and the current relational database. Further, because of the graph database, scaling the project in the future will be much easier. Also, like above, developers will notice an increase in performance making testing and maintenance much more responsive and much faster.

6. QUALIFICATIONS



Colin Schutte

Computer Science Student

About

I am currently a Senior in the Computer Science and Spanish programs at Montana State University

Personal

Aside from being a student at MSU, I am an avid fly fisher, hunter and snow-boarder

One thing that sets me apart from the crowd is my ability to learn very quickly and be able to apply what I have learned at the same pace. Before coming to college, I had never written a single program. I went through some initial struggles and had to work hard, but now would say I am quite proficient and am about to graduate school with a 3.8 GPA.

Education

Chaparral High School (2015-2019)
I graduated from Chaprral High School in 2019 with a 3.92 GPA. I played football all 4 years and was selected as Senior Captain by my teammates and coaches.

Montana State Univeristy (2019 -)
I accepted a competitive WUE scholarship and decided to come to MSU in 2019. I am pursuing a BA in Computer Science with a minor in Spanish.

Technical Experience

Programming Langauges
- Java
- Python
- C
- HTML/CSS
- JavaScript/TypeScript

Databases
-SQL
-Queries
-DDL
-GraphQL

Network Programming
-ORM
-CRUD Applications
-Socket Programming
-RESTFUL APIs
-OSI Model and Protocols

Software Design
-Various Types of UML Diagrams
-ERD Diagrams
-Various Software Design Patterns

Interests

As a result of my studies, my primary interests in the field of Computer Science are security and data. As we move further and further into the digital age, I look forward to learning about how these two fields coexist as they become more important and there becomes a higher need for more securi-ty as more data becomes available.

**Github and References Available on Request

Email:
colinshoots@gmail.com
Phone Number:
720-771-8237

Gregory Hill

Bozeman, MT | (406) 670-6864 | gregoryhill.mt@gmail.com

Objective

Computer Science student seeking a professional role, internship, and/or co-op to grow my skills.

Education

Bachelor of Science | August 2019 – May 2023 | Montana State University, Bozeman, MT

• GPA: 3.81, Computer Science Major, Honors Baccalaureate, Physics Minor

Relevant Projects

Senior Capstone Project (current) | CSCI 482R Interdisciplinary Project Instruction | Client: F5

• Research/develop a proof-of-concept/demo that integrates GraphQL with existing F5 products
• Reduce the impact of API sprawl
• Implement database-like queries for APIs

Machine Learning Projects (current) | CSCI 447 Machine Learning

• Four projects over the semester implementing learning models and algorithms
• Partner-based projects ranging from implementing linear models to neural networks

Compiler Project | CSCI 486 Compilers

• Completed implementation of a small compiler
• Utilized Java to implement parsing, transpiling, and bytecode generation

Robotics Project | CSCI 455 Embedded Systems

• Created a virtual board game played by the robot
• Implemented Python code to control the movement of the robot and the robot's arm, a GUI interface for interaction, and text-to-speech for player decisions

Experience

Lawn Service | May 2021 – August 2022 | Advantage Landcare, Bozeman MT

• Complete a set number of accounts as a team in a timely manner
• Ensure all lawns comply with the company and client's standards

Pool maintenance Specialist | June 2017 – August 2020 | Five Star Pool and Spa, Billings, MT

• Maintain the chemical balance of pool water
• Ensure all pool equipment functions properly
• Assist in the repair of pool equipment, installation of vinyl liners, and construction of pools

Sales Associate | June 2016 – June 2017 | Office Depot, Billings, MT

• Assist customers in choosing appropriate merchandise, especially for electronic products
• Cashier

Honors

• President's List (Fall 2019 and 2020) • Dean's List (Spring 2020, Spring 2021, and Spring 2022)
• Computer Science Scholarship Endowment (2020 – 2021) • Paxton Family Computer Science Undergraduate Student Success Scholarship (2021 – 2022) • Montana University System Scholarship (2019)
• Montana Chapter NECA Scholarship (2019, 2020) • Home Builders Association of Billings Scholars (2019)
• Billings Society of Petroleum Engineers Scholarship (2019)

Riley Williams

Riley Williams
3315 S. 27th
Bozeman, MT 59718

406-531-6219
riley.b.williams05@gmail.com

Skills

Experience with multiple coding languages, Java being the primary. I have experience successfully learning and applying new languages and have experience working with others to accomplish tasks on time.

Education

Montana State University / Computer Science Department, expected graduation - Spring 2023
2015- 2021-Present, Bozeman MT

GPA - 3.71 in Computer Science classes

Excel in Computer Systems, Data Structures and Algorithms, and Concepts/Programming Languages classes to pursue a Bachelor degree in Computer Science. Have completed requirements to obtain a Minor in Mathematics.

- Proficient in Java and familiar with Python
- Completed and presented group projects on time
- Complete assignments and projects to meet required deadlines

Florence Carlton High School / High School Diploma
2004 - 2008, Florence MT

Graduated with honors. Took and enjoyed advanced Math classes as electives.

Experience

Rooks Games and More / Assistant Manager
Jul 2018 - PRESENT, Bozeman MT

Responsible for ensuring Rooks Games and More runs efficiently, and creating positive experiences for all customers.

- Responsible for opening and closing processes and procedures
- Complete customer transactions efficiently and in a timely manner
- Interface with customers to ensure a variety of needs are met

University Catering / Student Manager
2010 - 2013 and 2015-2019, Bozeman MT

Supervised and directed staff to ensure guests had an excellent experience at events.

- Managed and efficiently resolved problems
- Supervised and directed staff
- Effectively managed staff at events with hundreds of guests

Awards & Achievements

Achieved the rank of Eagle Scout in the Boy Scouts of America; 2008

References

(1) “What Are Bot Attacks? Bot Mitigation for Web Apps & Apis.” *Signal Sciences*, Fastly, https://www.signalsciences.com/glossary/bot-attack-protection/.

(2) “ECMA-404.” *Ecma International*, 4 Feb. 2021, https://www.ecma-international.org/publications-and-standards/standards/ecma-404/.

(3) “Queries and Mutations.” *GraphQL*, https://graphql.org/learn/queries/.

(4) Microsoft Contributors. “The Api Gateway Pattern versus the Direct Client-to-Microservice Communication.” *The API Gateway Pattern versus the Direct Client-to-Microservice Communication | Microsoft Learn*, 21 Sept. 2022, https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/direct-client-to-microservice-communication-versus-the-api-gateway-pattern.

(5) “The Leader in Graph Databases.” *Neo4j Graph Data Platform*, 17 Nov. 2022, https://neo4j.com/.

(6) Ng'ethe, Joe. “Wrapping a REST API in GraphQL.” *Medium*, TwigaTech, 7 June 2018, https://medium.com/twigatech/wrapping-a-rest-api-in-graphql-4e26c6582282.

Appendix - Source Code

File Structure

- Tms-graph
    - .idea
    - .next
    - Apollo
        - Client.ts
        - schema.ts
        - typedefs
            - Config.ts
            - Customer.ts
            - Index.ts
            - Report.ts
            - Neo4jDirectives.ts
            - Test.ts
    - Components
    - Dev.sql
    - Node_modules
    - Pages
    - Public
    - Styles
    - typeORM
        - Datasources
            - Mysql.datasource.ts
            - neo4j.datasource.ts
        - Entity
            - customer.ts
            - integrationTest.ts
            - labels.ts
            - mobileBaseConfig.ts
            - orcaCustomer.ts
            - orcaTest.ts
            - scheduledTest.ts
            - siteAnalysis.ts
            - status.ts
            - test.ts
            - testSuite.ts
        - Migrations
            - dataMigration
            - testMigration

    - Types
        - apiTypes
        - Index.ts
    - Util
        - Bootstrap
            - generatePreprocessingFunctions.ts
            - generateTypes
            - preprocessorCodegen
            - utils.ts
        - tms_asg_codegen
            - preprocessorFunctions.ts
            - visitor.ts
        - neo4j.ts
        - tsconfig.json
    - .env.local
    - .gitignore
    - docker-compose.yaml
    - next.config.js
    - next-env.d.ts
    - package.json
    - README.md
    - tsconfig.json
    - Yarn.lock
    - yarn-error.log

Source Code

apollo/typedefs/

config.ts

```
import {gql} from '@apollo/client';
import {gql} from '@apollo/client';

export const typeDefs = gql`
  type OrcaTestConfig {
     id: ID! @id
     isTestSuiteUsed: Boolean!
     browserSmokeTests: BrowserSmokeTests! @relationship(type:
"HAS_BROWSER_SMOKE_TEST_OF_KIND_BROWSER_SMOKE_TESTS", direction: OUT)
     dnsSpoof: DnsSpoof! @relationship(type: "HAS_DNS_SPOOF_OF_KIND_DNS_SPOOF",
direction: OUT)
     integrationTest: IntegrationTest! @relationship(type:
"HAS_INTEGRATION_TEST_OF_KIND_INTEGRATION_TEST", direction: OUT)
```

```
    functionalMobileTest: FunctionalMobileTest! @relationship(type:
"HAS_FUNCTIONAL_MOBILE_TEST_OF_KIND_FUNCTIONAL_MOBILE_TESTS", direction: OUT)
    functionalTest: FunctionalTest! @relationship(type:
"HAS_FUNCTIONAL_TEST_OF_KIND_FUNCTIONAL_TESTS", direction: OUT)
    functionalWebTest: FunctionalWebTest! @relationship (type:
"HAS_FUNCTIONAL_WEB_TEST_OF_KIND_FUNCTIONAL_WEB_TESTS", direction: OUT)
    fingerprint: String!
  }
  type FunctionalWebBrowserConfiguration {
    id: ID! @id
    os: OS!
    osVersion: String!
    browser: Browser!
    browserVersion: String!
    fingerprint: String!

  }
  type MobileDeviceConfiguration {
    id: ID! @id
    os: OS!
    osVersion: String!
    device: String!
    realMobile: String!
    fingerprint: String!
  }
  type MobileBaseConfig {
    id: ID! @id
    #jobMeta: JobMeta! @relationship(type: "HAS_JOB_META_OF_KIND_JOB_META", direction:
OUT)
    createdOn: DateTime!
    createdBy: String!
    finishedOn: DateTime!
    updatedOn: DateTime!
    customerName: String!
    jobName: String!
    beh: String
    net: String
    ti: String
    tiName: String!
    cluster: String
    merged: String
    pcUrl: String!
    result: String
    messages: [Message!]! @relationship(type: "HAS_MESSAGES_OF_KIND_MESSAGE",
direction: OUT)
    artifacts: [Artifact!]! @relationship(type: "HAS_ARTIFACTS_OF_KIND_ARTIFACTS", direction:
OUT)
    counter: Int!
    pegasusGcrUrl: String
    policyName: String!
    #customer: OrcaCustomerLite! @relationship(type:
"HAS_CUSTOMER_OF_KIND_ORCA_CUSTOMER_LITE", direction: IN)
    #policy: PolicyV1! @relationship(type: "HAS_POLICY_OF_KIND_POLICY_V_1", direction: IN)
    firmware: String!
    #mobileConfig: [MobileConfig!]! @relationship(type:
"HAS_MOBILE_CONFIG_OF_KIND_MOBILE_CONFIG", direction: OUT)
```

```
    status: String!
    fingerprint: String!

  }
  type MobileConfig {
    id: ID! @id
    sdkVersion: SupportedSdkVersion! @relationship(type:
"HAS_SDK_VERSION_OF_KIND_SUPPORTED_SDK_VERSION", direction: OUT)
    sdkType: String!
    os: OS!
    environment: String!
    fingerprint: String!
  }
  type SupportedSdkVersion {
    id: ID! @id
    sdkVersion: String!
    fingerprint: String!
  }
  type SlackNotifConfig {
    id: ID! @id
    notifyOnFail: Boolean!
    notifyOnPass: Boolean!
    testDetails: TestDetail! @relationship(type: "HAS_TEST_DETAILS_OF_KIND_TEST_DETAIL",
direction: OUT)
    pingUsers: [String!]!
    fingerprint: String!
  }
  type TestDetail {
    id: ID! @id
    firmware: Boolean!
    enterprise: Boolean!
    policyName: Boolean!
    ti: Boolean!
    networkPolicyName: Boolean!
    pcURL: Boolean!
    createdOn: Boolean!
    createdBy: Boolean!
    environment: Boolean!
    finishedOn: Boolean!
    fingerprint: String!
  }
  type DnsSpoof {
    id: ID! @id
    domain: String!
    ip: String!
    fingerprint: String!
  }
  type Artifact {
    id: ID! @id
    url: String!
    createdOn: DateTime!
    fingerprint: String!
  }

  enum OS {
    WINDOWS
    MACOS
    LINUX
    CHROMEOS
    IOS
    ANDROID
  }

  enum Browser {
    CHROME
    FIREFOX
    EDGE
  }

  type Endpoint {
    id: ID! @id
    saEndpoint: SiteAnalysisEndpoint @relationship(type:
"HAS_SA_ENDPOINT_OF_KIND_SITE_ANALYSIS_ENDPOINT", direction: OUT)
    name: String
    requestMatcher: String
    flowLabel: String
    platform: Platform
    protocol: Protocol
    url: String
    method: Method
    body: String
    curl: String
    queryParams: [String]
    headers: [HeaderMap]
    scriptName: String
    fingerprint: String!
  }

  type SiteAnalysisEndpoint {
    id: ID! @id
    fingerprint: String!
  }

  enum Platform {
    ANDROID
    IOS
  }

  enum Protocol {
    TCP
    UDP
    SCTP
  }


  type HeaderMapEntries {
    id: ID! @id
    key: String!
    value: String!
    fingerprint: String!
  }
  type HeaderMap {
     id: ID! @id
     entries: [HeaderMapEntries!]! @relationship(type:
"HAS_ENTRIES_OF_KIND__HEADER_MAP_ENTRIES_", direction: OUT)
     fingerprint: String!
  }
`
```

customer.ts

```
`
type OrcaCustomer {
       id: ID! @id
       name: String!
       orcaService: OrcaService!
       customer: Customer! @relationship(type:
"HAS_CUSTOMER_OF_KIND_ORCA_CUSTOMER", direction: OUT) #eid
       eidOverride: String!
       policy: Policy! @relationship(type: "HAS_POLICY_OF_KIND_POLICY",
direction: OUT) #pcid
       seleniumImage: String!
       meta: EndpointMeta! @relationship(type:
"HAS_META_OF_KIND_ENDPOINT_META", direction: OUT)
       notesURL: String!
       proxy: String!
       ssdpTableName: String!
       mitmCcAllowDomainList: String!
       mitmCcAddHeaders: String!
       mitmCcSpoofIp: String!
       slackNotifConfigDefaults: SlackNotifConfigMap! @relationship(type:
"HAS_SLACK_NOTIF_CONFIG_DEFAULTS_OF_KIND_SLACK_NOTIF_CONFIG_MAP", direction:
OUT)
       fingerprint: String!
}
type SlackNotifConfigMap {
       id: ID! @id
       key: String!
       value: SlackNotifConfig! @relationship(type:
"HAS_VALUE_OF_KIND_SLACK_NOTIF_CONFIG", direction: OUT)
}
type OrcaCustomerLite {
       id: ID! @id
       name: String!
       customer: Customer! @relationship(type:
"HAS_CUSTOMER_OF_KIND_ORCA_CUSTOMER", direction: OUT) #eid
       seleniumImage: String!
       proxy: String!
```

```
        ssdpTableName: String!
        mitmCcAllowDomainList: String!
        mitmCcAddHeaders: String!
        mitmCcSpoofIp: String!
        fingerprint: String!
}
type Customer {
        id: ID! @id
        customerName: String
        cluster: String
        taasAvailability: Int
        proxy: String!
        fingerprint: String!
}
enum OrcaService{
        ENABLE
        DISABLE
}
union Policy = PolicyV0 | PolicyV1
type PolicyV0 {
        id: ID! @id
        network: String
        behavior: String
        ti: String
        merged: String
        mode: String
        pcVersion: String
        pcURL: String
        fingerprint: String!
}
type PolicyV1 {
        id: ID! @id
        network: String
        behavior: String
        ti: String
        tiPackageName: String
        merged: String
        pcURL: String
        pcNetworkName: String
        cluster: String
        fingerprint: String!
}
type EndpointMeta {
        id: ID! @id
        createdBy: String
        createdOn: DateTime
        updatedOn: DateTime
        updatedBy: String
        fingerprint: String!
```

```
}
```

report.ts

```
type Report {
   id: ID! @id
   functional: [FunctionalReport!]! @relationship(type:
"HAS_FUNCTIONAL_OF_KIND_FUNCTIONAL_REPORT", direction: OUT)
   integration: [IntegrationReport!]! @relationship(type:
"HAS_INTEGRATION_OF_KIND_INTEGRATION_REPORT", direction: OUT)
   functionalMobile: [FunctionalMobileReport!]! @relationship(type:
"HAS_FUNCTIONAL_MOBILE_OF_KIND_FUNCTIONAL_MOBILE_REPORT", direction: OUT)
   functionalWeb: [FunctionalWebReport!]! @relationship(type:
"HAS_FUNCTIONAL_WEB_OF_KIND_FUNCTIONAL_WEB_REPORT", direction: OUT)
   fingerprint: String!
}
type FunctionalMobileReport {
   id: ID! @id
   functionalMobileTest: FunctionalMobileTest! @relationship(type:
"HAS_FUNCTIONAL_MOBILE_TEST_OF_KIND_FUNCTIONAL_MOBILE_TEST", direction: OUT)
   resultCode: Int!
   results: [MobileResult!]! @relationship(type:
"HAS_RESULTS_OF_KIND_MOBILE_RESULT", direction: OUT)
   messages: [Message!]! @relationship(type: "HAS_MESSAGES_OF_KIND_MESSAGE",
direction: OUT)
   fingerprint: String!
}
type FunctionalReport {
   id: ID! @id
   resultCode: Int!
   results: [IntegrationResult!]! @relationship(type:
"HAS_RESULTS_OF_KIND_INTEGRATION_RESULT", direction: OUT)
   messages: [Message!]! @relationship(type: "HAS_MESSAGES_OF_KIND_MESSAGE",
direction: OUT)
   fingerprint: String!
}
type FunctionalWebReport {
   id: ID! @id
   resultCode: Int!
   results: [IntegrationResult!]! @relationship(type:
"HAS_RESULTS_OF_KIND_INTEGRATION_RESULT", direction: OUT)
   messages: [Message!]! @relationship(type: "HAS_MESSAGES_OF_KIND_MESSAGES",
direction: OUT)
   fingerprint: String!
}
```

```
type IntegrationReport {
   id: ID! @id
   resultCode: Int!
   results: [IntegrationResult!]! @relationship(type:
"HAS_RESULTS_OF_KIND_INTEGRATION_RESULT", direction: OUT)
   messages: [Message!]! @relationship(type: "HAS_MESSAGES_OF_KIND_MESSAGE",
direction: OUT)
   explanation: TestReportExplanation @relationship(type:
"HAS_EXPLANATION_OF_KIND_TEST_REPORT_EXPLANATION", direction: OUT)
   fingerprint: String!
}
type TestReportExplanation {
   name: String!
   scenario: String!
   osBrowser: String!
   field: String!
   operation: String!
   expectedValue: String!
   actualValue: String!
   fingerprint: String!
}
type Message {
   id: ID! @id
   severity: String!
   text: String!
   timeStamp: DateTime!
   fingerprint: String!
}
type IntegrationResult {
   id: ID! @id
   expectation: Expectation! @relationship(type:
"HAS_EXPECTATION_ID_OF_KIND_EXPECTATION", direction: OUT)
   assertion: Assertion! @relationship(type:
"HAS_ASSERTION_ID_OF_KIND_ASSERTION", direction: OUT)
   resultCode: Int!
   messages: [Message!]! @relationship(type: "HAS_MESSAGES_OF_KIND_MESSAGES",
direction: OUT)
   explanation: Explanation! @relationship(type:
"HAS_EXPLANATION_OF_KIND_EXPLANATION", direction: OUT)
   fingerprint: String!
}
type MobileResult {
   id: ID! @id
   scenarioId: String!
   expectation: Expectation! @relationship(type:
"HAS_EXPECTATION_OF_KIND_EXPECTATION", direction: OUT)
   assertion: Assertion! @relationship(type:
"HAS_ASSERTION_OF_KIND_ASSERTION", direction: OUT)
   resultCode: Int!
   explanation: Explanation! @relationship(type:
"HAS_EXPLANATION_OF_KIND_EXPLANATION", direction: OUT)
   messages: [Message!]! @relationship(type: "HAS_MESSAGES_OF_KIND_MESSAGES",
direction: OUT)
   fingerprint: String!
}

type Expectation {
   id: ID! @id
   type: String!
   name: String!
   value: Int!
   fingerprint: String!
}
type Assertion {
   id: ID! @id
   operation: String!
   value: [String!]!
   fingerprint: String!
}
type Explanation {
   id: ID! @id
   name: String!
   scenario: String!
   osBrowser: String!
   field: String!
   operation: String!
   expectedValue: String!
   actualValue: String!
   fingerprint: String!
}
```

test.ts

```
type OrcaTest {
       id: ID! @id
       createdOn: DateTime
       createdBy: String
       finishedOn: DateTime
       name: String
       jobMeta: JobMeta @relationship(type: "HAS_JOB_META_OF_KIND_JOB_META",
direction: OUT)
       customer: OrcaCustomerLite @relationship(type:
"HAS_CUSTOMER_OF_KIND_ORCA_CUSTOMER_LITE", direction: OUT)
       policy: PolicyV1 @relationship(type: "HAS_POLICY_OF_KIND_POLICY_V_1",
direction: OUT)
```

```
	firmware: FirmwareV1 @relationship(type:
"HAS_FIRMWARE_OF_KIND_FIRMWARE_V_1", direction: OUT)
	compileOnly: Boolean!
	testInProd: Boolean!
	useXDebugTag: Boolean!
	useSpoofingIp: Boolean!
	testConfig: OrcaTestConfig!
	customerName: String
	status: Status @relationship(type: "HAS_STATUS_OF_KIND_STATUS",
direction: OUT)
	proxy: String
	slackNotificationConfig: SlackNotifConfig! @relationship(type:
"HAS_SLACK_NOTIFICATION_CONFIG_OF_KIND_SLACK_NOTIF_CONFIG", direction: OUT)
	fingerprint: String!
}
type ScheduledTest {
	id: ID! @id
	createdOn: DateTime
	createdBy: String
	name: String
	jobMeta: JobMeta @relationship(type: "HAS_JOB_META_OF_KIND_JOB_META",
direction: OUT)
	customer: OrcaCustomerLite @relationship(type:
"HAS_CUSTOMER_OF_KIND_ORCA_CUSTOMER_LITE", direction: OUT)
	policy: PolicyV1 @relationship(type: "HAS_POLICY_OF_KIND_POLICY_V_1",
direction: OUT)
	firmware: FirmwareV1 @relationship(type:
"HAS_FIRMWARE_OF_KIND_FIRMWARE_V_1", direction: OUT)
	compileOnly: Boolean!
	testInProd: Boolean!
	useXDebugTag: Boolean!
	useSpoofingIp: Boolean!
	testConfig: OrcaTestConfig! @relationship(type:
"HAS_TEST_CONFIG_OF_KIND_ORCA_TEST_CONFIG", direction: OUT)
	proxy: String
	startOn: DateTime
	lastRanOn: DateTime
	runInterval: String
	endOn: DateTime
	numRuns: Int!
	maxRuns: Int
	slackNotificationConfig: SlackNotifConfig! @relationship(type:
"HAS_SLACK_NOTIFICATION_CONFIG_OF_KIND_SLACK_NOTIF_CONFIG", direction: OUT)
	fingerprint: String!
}
type Test {
	id: ID! @id
	customer: Customer! @relationship(type:
"HAS_CUSTOMER_OF_KIND_CUSTOMER", direction: OUT)
```

```
        policy: Policy! @relationship(type: "HAS_POLICY_OF_KIND_POLICY",
direction: OUT)
        firmware: Firmware! @relationship(type:
"HAS_FIRMWARE_OF_KIND_FIRMWARE", direction: OUT)
        browsers: [String!]!
        type: [String!]!
        testName: String
        createdOn: DateTime!
        conf: ConfigMapping! @relationship(type:
"HAS_CONF_OF_KIND_CONFIG_MAPPING", direction: OUT)
        fingerprint: String!
}
type ConfigMapping {
        id: ID! @id
        key: String!
        webRegressionTests: [WebRegressionTest!]! @relationship(type:
"HAS_WEB_REGRESSION_TESTS_OF_KIND_WEB_REGRESSION_TEST", direction: OUT)
        fingerprint: String!
}
type WebRegressionTest {
        id: ID! @id
        endpoint: Endpoint @relationship(type: "HAS_ENDPOINT_OF_KIND_ENDPOINT",
direction: OUT)
        entryPoint: String
        flowLabel: String
        method: Method
        name: String
        requestMatcher: String
        bodyMatcher: [String!]
        headerMatcher: [String!]
        queryParams: [String!]
        status: TestStatus! @relationship(type:
"HAS_STATUS_OF_KIND_TEST_STATUS", direction: OUT)
        expectations: [Expectation!]! @relationship(type:
"HAS_EXPECTATIONS_OF_KIND_EXPECTATION_", direction: OUT)
        fingerprint: String!
}

type TestStatus {
        id: ID! @id
        integration: Int!
        functional: Int!
        fingerprint: String!
}

enum Method {
        GET
        HEAD
        POST
```

```
       PUT
       DELETE
       CONNECT
       OPTIONS
       TRACE
       PATCH
}
union Firmware = FirmwareV0 | FirmwareV1
type FirmwareV0 {
       id: ID! @id
       sse: String
       pegasusURL: String
       fingerprint: String!
}
type FirmwareV1 {
       id: ID! @id
       sseFirmware: String
       gcrPegasusURL: String
       fingerprint: String!
}
type BrowserSmokeTests {
       id: ID! @id
       urls: [String]!
       shapeClientJsPaths: [String]!
       infraConfigs: [InfraConfig!]! @relationship(type:
"HAS_CONFIG_OF_KIND_INFRA_CONFIG", direction: OUT)
       fingerprint: String!
}

type InfraConfig {
       id: ID! @id
       os: String!
       osVersion: String!
       BrowserName: String!
       BrowserVersion: String!
       SeleniumVersion: String!
       fingerprint: String!
}
type TestSuite {
       id: ID! @id
       orcaCustomer: OrcaCustomer! @relationship(type:
"HAS_ORCA_CUSTOMER_OF_KIND_ORCA_CUSTOMER", direction: OUT) #oeid
       setAsDefault: Boolean
       meta: EndpointMeta! @relationship(type:
"HAS_META_OF_KIND_ENDPOINT_META", direction: OUT)
       tests: [EncodedTest!]! @relationship(type:
"HAS_TESTS_OF_KIND_ENCODED_TEST", direction: OUT)
       fingerprint: String!
}
```

```
type EncodedTest {
       id: ID! @id
       testType: TestType!
       policyName: String!
       labels: [String!]!
       isArchived: Boolean!
       test: Test @relationship(type: "HAS_TEST_OF_KIND_TEST", direction: OUT)
       fingerprint: String!
}
enum TestType {
       INTEGRATION
       FUNCTIONAL_MOBILE
       FUNCTIONAL
       FUNCTIONAL_WEB
}
type JobMeta {
       id: ID! @id
       policyName: String
       isScheduledTest: Boolean!
       fingerprint: String!
}
type Status {
       id: ID! @id
       logs: String!
       message: String!
       result: String!
       status: String!
       testName: String!
       createdOn: DateTime!
       customerName: String!
       integrationResult: String!
       functionalResult: String!
       fingerprint: String!
}

type Label {
       id: ID! @id
       orcaTest: OrcaTest! @relationship(type:
"HAS_ORCA_TEST_OF_KIND_ORCA_TEST", direction: OUT)
       labels: [String]
       createdOn: DateTime
       updatedOn: DateTime
       fingerprint: String!
}
type FunctionalWebTest {
       id: ID! @id
       version: String!
       ids: [String!]!
```

```
	browserConfigs: [FunctionalWebBrowserConfiguration!]!
@relationship(type:
"HAS_BROWSER_CONFIGS_OF_KIND_FUNCTIONAL_WEB_BROWSER_CONFIGURATION_",
direction: OUT)
	fingerprint: String!
}

union FunctionalMobileTest = FunctionalMobileTestV1 | FunctionalMobileTestV2

type FunctionalMobileTestV1 {
	id: ID! @id
	sdkVersion: String!
	Version: String!
	ids: [String!]!
	updateUrlKey: String!
	bundleMatcher: String!
	deviceConfigs: [MobileDeviceConfiguration!]! @relationship(type:
"HAS_DEVICE_CONFIGS_OF_KIND_MOBILE_DEVICE_CONFIGURATION_", direction: OUT)
	platform: Platform!
	fingerprint: String!
}


type FunctionalMobileTestV2 {
	id: ID! @id
	endpoint: Endpoint @relationship(type: "HAS_ENDPOINT_OF_KIND_ENDPOINT",
direction: OUT)
	expectations: String
	fingerprint: String!
}

union FunctionalTest = FunctionalTestV1 | FunctionalTestV2

type FunctionalTestV1 {
	id: ID! @id
	version: String!
	ids: [String!]!
	seleniumIds: [String]!
	fingerprint: String!
}

type FunctionalTestV2 {
	id: ID! @id
	name: String
	endpoint: Endpoint @relationship(type: "HAS_ENDPOINT_OF_KIND_ENDPOINT",
direction: OUT)
	entrypoint: String
	requestMatcher: String
	flowLabel: String
	fingerprint: String!
}

type GenericTest {
	id: ID! @id
	endpoint: Endpoint! @relationship(type:
"HAS_ENDPOINT_OF_KIND_ENDPOINT", direction: OUT)
	expectations: [String!]! # struct has json.RawMessage which is a byte
array, not sure how best to store
	fingerprint: String!
}

union IntegrationTest = IntegrationTestV1 | IntegrationTestV2

type IntegrationTestV1 {
	id: ID! @id
	orcaCustomer: OrcaCustomer! @relationship(type:
"HAS_ORCA_CUSTOMER_OF_KIND_ORCA_CUSTOMER", direction: OUT)
	setAsDefault: Boolean
	meta: EndpointMeta! @relationship(type:
"HAS_META_OF_KIND_ENDPOINT_META", direction: OUT)
	tests: [GenericTest!]! @relationship(type:
"HAS_TESTS_OF_KIND_GENERIC_TEST_", direction: OUT)
	fingerprint: String!

}

type IntegrationTestV2 {
	id: ID! @id
	version: String!
	ids: [String]
	fingerprint: String!
}
```

<u>typeORM/datasources</u>
mysql.datasource.ts

```
import { DataSource } from "typeorm"
import { Customer } from "../entity/customer";
import { Integrationtest } from "../entity/integrationTest";
import { Labels } from "../entity/labels";
import { Mobilebaseconfig } from "../entity/mobileBaseConfig";
import { Orcacustomer } from "../entity/orcaCustomer";
import { Orcatest } from "../entity/orcaTest";
import { Siteanalysis } from "../entity/siteAnalysis";
import { Status } from "../entity/status";
import { Test } from "../entity/test";
import { Testsuite } from "../entity/testSuite";
import "reflect-metadata"

export const MySQLDataSource = new DataSource({
   type: "mysql",
   host: "localhost",
   port: 3306,
   username: "orca",
   password: "changeme",
   database: "dorsal_fin_db",
   entities: [Customer, Integrationtest, Orcacustomer, Labels,
Mobilebaseconfig, Orcatest, Siteanalysis, Status, Test, Testsuite]
})
```

neo4j.datasource.ts

```
import neo4j, { Driver } from 'neo4j-driver';

export class Neo4jDataSource {
   private driver: Driver;
   constructor() {
       this.connect();
   }

   connect() {
       this.driver = neo4j.driver(
           'bolt://localhost:7687',
           neo4j.auth.basic('username', 'password'),
       );
   }

   getDriver() {
       return this.driver;
   }

   async close() {
       await this.driver.close();
   }
}
```

<u>typeORM/entity</u>
customer.ts

```
import { Entity, PrimaryColumn, Column } from "typeorm"

@Entity()
export class Customer {
   // @PrimaryColumn()
   // id: string
   @PrimaryColumn()
   eid: string

   @Column()
   cluster: string

   @Column()
   customer: string

   @Column()
   taaS: boolean

   /*@Column()
   proxy: string*/

  /* @Column()
   ssdpTableName: string*/

  /* @Column()
   mitmCcAllowDomainList: string*/

  /* @Column()
   mitmCcAddHeaders: string*/

  /* @Column()
   mitmCcSpoofIp: string*/
```

integrationTest.ts

```
import {
   Entity,
   PrimaryGeneratedColumn,
   PrimaryColumn,
   Column,
   ManyToOne, OneToOne, JoinColumn
} from "typeorm"
import "reflect-metadata"
import { Orcacustomer } from "./orcaCustomer"
import {DateTime} from "neo4j-driver";


@Entity()
export class Integrationtest {
   @PrimaryColumn()
   uuid: string

   @OneToOne(() => Orcacustomer, (orcaCustomer) => orcaCustomer.oEid)
   oEid: Orcacustomer
   @OneToOne(() => Orcacustomer, (orcaCustomer) => orcaCustomer.eid)
  // @JoinColumn
   eid: Orcacustomer

   @Column("json")
   data: string

   @Column()
   isDefault: boolean

   @Column()
   version: number

   @PrimaryColumn("bigint")
   counter: number

   @Column("datetime")
   createdOn: string

   @Column()
   createdBy: string

   @Column("datetime")
   updatedOn: string

   @Column()
   updatedBy: string
}
```

labels.ts

```
import {Entity, PrimaryGeneratedColumn, PrimaryColumn, Column, OneToMany,
Generated, ManyToOne, OneToOne, JoinColumn} from "typeorm"
import {Orcacustomer} from "./orcaCustomer"

@Entity()
export class Labels {
   // @PrimaryGeneratedColumn()
   // id

   @PrimaryColumn()
   labelId: string

   /*@OneToOne(() => Orcacustomer, (orcaCustomer) => orcaCustomer.oEid)
   @JoinColumn()
   oEid: Orcacustomer*/

   @Column()
   oEid: string

   @Column('json')
   data: string

   @Column('datetime')
   createdOn: Date

   @Column('datetime')
   updatedOn: Date

   @Column("bigint")
   counter: number

   // @Column()
   // counter: bigint

}
```

mobileBaseConfig.ts

```
import {Entity, PrimaryGeneratedColumn, Column, PrimaryColumn, ManyToOne,
OneToOne} from "typeorm";
import { Orcacustomer } from "./orcaCustomer"
import {DateTime} from "neo4j-driver";
import { Message } from "../../types/apiTypes";

@Entity()

export class Mobilebaseconfig {
   @PrimaryColumn()
   jobId: string
   @OneToOne(() => Orcacustomer,(orcaCustomer) => orcaCustomer.eid)
   eid: string
   @Column()
   customerName: string
   @Column()
   jobName: string
   @Column({type: "datetime"})
   createdOn: DateTime
   @Column()
   createdBy: string
   @Column({type: "datetime"})
   finishedOn: DateTime
   @Column({type: "datetime"})
   updatedOn: DateTime
   @Column({type: "mediumtext"})
   beh: string
   @Column({type: "mediumtext"})
   net: string
   @Column({type: "mediumtext"})
   ti: string
   @Column()
   tiName: string
   @Column()
   cluster: string
   @Column({type: "mediumtext"})
   merged: string
   @Column()
   pcUrl:string
   @Column({type: "json"})
   jobConfig: any
   @Column()
   status: string
   @Column()
   result: string
   @Column("json")
   messages: string
   @Column("json")
   artifacts: string
   @PrimaryColumn({type: "bigint"})
   counter: bigint
   @Column()
   firmware: string
   @Column()
   pegasusGcrUrl: string
   @Column()
   policyName: string

}
```

orcaCustomer.ts

```
import {
   Entity,
   PrimaryGeneratedColumn,
   PrimaryColumn,
   Column,
   ManyToOne
} from "typeorm"


@Entity()
export class Orcacustomer {

   @PrimaryColumn()
   oEid: string

```

```
  @Column()
  eid: string

 /* @Column()
  eidOverride: string*/

  /*@Column()
  pcId: string*/

  @Column()
  seleniumImage: string

  @Column()
  name: string

  @Column()
  taaS: string

  @Column("datetime")
  createdOn: string

  @Column()
  createdBy: string

  @Column("datetime")
  updatedOn: string

  @Column()
  updatedBy: string

  @Column()
  pcId: string

  @Column()
  notesUrl: string

 /* @Column()
  proxy: string*/

  /*@Column()
  ssdpTableName: string*/

 /* @Column()
  mitmCcAllowDomainList: string*/

 /* @Column()
  mitmCcAddHeaders: string*/

  /*@Column()
```

```
   mitmCcSpoofIp: string*/

   /*@Column()
   slackNotificationConfigDefaults: string*/
}
```

orcaTest.ts

```
import {Column, PrimaryColumn, Entity, ManyToOne} from "typeorm";
import {Orcacustomer} from "./orcaCustomer";
import {DateTime} from "neo4j-driver";

@Entity()
export class Orcatest {
   @PrimaryColumn()
   testId: string
  /* @ManyToOne(() => Orcacustomer,(orcaCustomer) => orcaCustomer.eid)*/
   @Column()
   eid: string
   /*@Column()
   oEid: string*/
   @Column()
   customerName: string
   @Column()
   testName: string
   @Column({type: "datetime"})
   createdOn: DateTime
   @Column()
   createdBy: string
   @Column({type: "datetime"})
   finishedOn: DateTime
   @Column({type: "datetime"})
   updatedOn: DateTime
   @Column({type: "mediumtext"})
   beh: string
   @Column({type: "mediumtext"})
   net: string
   @Column({type: "mediumtext"})
   ti: string
   @Column()
   tiName:string
   @Column()
   cluster:string
   @Column()
   merged: string
   @Column()
   pcUrl:string
   @Column({type:"json"})
   testConfig: any
   @Column()
   status: string
   @Column()
   result: string
   @Column({type: "json"})
   messages: any
   @Column({type: "json"})
   artifacts: any
   @PrimaryColumn({type: "bigint"})
   counter: bigint
   @Column()
   firmware: string
   @Column()
   pegasusGcrUrl:string
   @Column({type:"json"})
   integrationReport: any
   @Column({type:"json"})
   functionalReport: any
   @Column()
   oEid: string
   @Column()
   policyName: string
   @Column({type:"json"})
   mobileReport: any
   @Column()
   compileOnly: boolean
   @Column()
   pcNetworkName: string
   @Column({type:"json"})
   functionalWebReport: any
   /*@Column()
   compileOnly: boolean*/
  /* @Column()
   testInProd: boolean*/
  /* @Column()
   useXDebugTag: boolean*/
  /* @Column()
   useSpoofingIp: boolean*/
   /*@Column()
   isScheduledTest: boolean*/
  /* @Column()
   pcNetworkName: string*/
   /*@Column()
   customerProxy: string*/
  /* @Column()

   slackNotificationConfig: string*/
}
```

scheduledTest.ts

```
import {
   Entity,
   PrimaryGeneratedColumn,
   PrimaryColumn,
   Column,
   ManyToOne
} from "typeorm";
import {DateTime} from "neo4j-driver";

@Entity()
export class ScheduledTest {
   @PrimaryColumn()
   scheduledTestId: string

   @ManyToOne(() => Orcacustomer, (eid) => orcaCustomer.eid)
   eid: string

   @Column()
   oEid: string

   @Column()
   customerName: string

   @Column()
   testName: string

   @Column()
   createdOn: DateTime

   @Column()
   createdBy: string

   @Column()
   updatedOn: DateTime

   @Column()
   beh: string

   @Column()
   net: string

   @Column()
   ti: string

   @Column()
   tiName: string
```

```
@Column()
cluster: string

@Column()
merged: string

@Column()
pcUrl: string

@Column()
testConfig: any

@Column()
firmware: string

@Column()
pegasusGcrUrl: string

@Column()
policyName: string

@Column()
compileOnly: boolean

@Column()
testInProd: boolean

@Column()
useXDebugTag: boolean

@Column()
useSpoofingIp: boolean

@Column()
pcNetworkName: string

@Column()
startOn: DateTime

@Column()
lastRanOn: DateTime

@Column()
runInterval: string

@Column()
endOn: DateTime

@Column()
```

```
   numRuns: number

   @Column()
   MaxRuns: number

   @Column()
   slackNotificationConfig: string
}
```

siteAnalysis.ts

```
import { Entity, PrimaryGeneratedColumn, PrimaryColumn, Column, ManyToOne,
Generated } from "typeorm"
import {DateTime} from "neo4j-driver";
import {Orcacustomer} from "./orcaCustomer";

@Entity()
export class Siteanalysis {
   /*@PrimaryGeneratedColumn()
   id*/

   //primary key?
   @PrimaryColumn({nullable: false})
   saId: string

 /*  @Column({nullable: false})
   oEid: string*/

   /*@ManyToOne(() => Orcacustomer, (orcaCustomer) => orcaCustomer.oEid)*/
   @Column()
   oEid: string

   @Column()
   eid: string

   @Column('json')
   data: any

   @Column({nullable: false})
   isDefault: boolean

   @Column({nullable: false})
   version: number

   @Column("bigint")
  /* @Generated('increment')*/
   counter: string

   @Column('datetime')
```

```
    createdOn: Date

    @Column()
    createdBy: string

    @Column('datetime')
    updatedOn: Date

    @Column()
    updatedBy: string
}
```

status.ts

```
import { Entity, PrimaryGeneratedColumn, PrimaryColumn, Column, OneToMany,
Generated } from "typeorm"
import {DateTime} from "neo4j-driver";

@Entity()
export class Status {
    /*@PrimaryGeneratedColumn()
    id*/

    @PrimaryColumn({nullable: false})
    testId: string

    @Column()
    logs: string

    @Column()
    message: string

    @Column()
    result: string

    @Column({nullable: false})
    status: string


    @Column()
    testName: string

    @Column('datetime')
    createdOn: Date

    @Column()
    customerName: string

    @Column()
```

```
   integration_result: string

   @Column()
   functional_result: string
}
```

test.ts

```
import {
   Entity,
   PrimaryGeneratedColumn,
   PrimaryColumn,
   Column,
   ManyToOne,
   OneToOne,
   JoinColumn
} from "typeorm"
import {DateTime} from "neo4j-driver";
import {Status} from "./status";
import {Customer} from "./customer";

@Entity()
export class Test {
   @PrimaryColumn()
   testId: string

  /* @OneToOne(() => Status)
   @JoinColumn()
   testId: Status*/

   /*@ManyToOne(() => Customer, (customer) => customer.eid)
   eid: Customer*/

   @Column()
   eid: string

   @Column()
   beh: string

   @Column()
   net: string

   @Column()
   ti: string

   @Column()
   merged: string

   @Column()
   browsers: string

   @Column()
   cluster: string

   @Column()
   cname: string

   @Column()
   firmware: string

   @Column()
   policyMode: string

   @Column()
   pegasusGUrl: string

   @Column()
   testType: string

   @PrimaryColumn()
   counter: number

   @Column()
   testName: string

   @Column("datetime")
   createdOn: Date

   @Column()
   pcId: string

   @Column()
   pcVersion: string

   @Column()
   pcUrl: string

   @Column()
   config: string

  /* @Column()
   proxy: string*/

   /*@Column()
   ssdpTableName: string*/

  /* @Column()
   mitmCcAllowDomainList: string*/
  /* @Column()
   mitmCcAddHeaders: string*/

   /*@Column()
   mitmCcSpoofIp: string*/
}
```

testSuite.ts

```
import {Entity, Column, PrimaryColumn, ManyToOne} from "typeorm";
import {Orcacustomer} from "./orcaCustomer";
import {DateTime} from "neo4j-driver";
@Entity()
export class Testsuite{
   @PrimaryColumn()
   TSid: string
   /*@ManyToOne(() => Orcacustomer,(orcaCustomer) => orcaCustomer.oEid)
   oEid: string*/
   @Column()
   oEid: string

   @Column()
   eid: string
   @Column({type: "json"})
   data: any
   @Column({type: "json"})
   testRunMeta: any
   @Column()
   isDefault: boolean
   @Column({type: "int"})
   version: number
   @PrimaryColumn({type: "bigint"})
   counter: string
   @Column({type: "datetime"})
   createdOn: DateTime
   @Column()
   createdBy:string
   @Column({type: "datetime"})
   updatedOn: DateTime
   @Column()
   updatedBy:string
}
```

<u>**typeORM/migrations**</u>
data_migration.ts

```
import { MySQLDataSource } from "../datasources/mysql.datasource"
import process from "process";
import {Context} from "../../types";
import { ogm } from "../../apollo/schema"
import neo4j from "neo4j-driver";
import { Customer } from "../entity/customer"
import { NodeMap } from "../../types";
import { preprocessCustomer } from
"../../util/tms_asg_codegen/preprocessorFunctions";
import { Mobilebaseconfig } from "../entity/mobileBaseConfig";
import {Artifact, JobMeta, Message} from "../../types/apiTypes";

const driver = neo4j.driver(
   "bolt://localhost:7687",
   neo4j.auth.basic("neo4j", "password")
);
const getContext = async():Promise<Context>=>{
   let session = driver.session()
   let updateFingerprintCacheSession = driver.session()
   await ogm.init()
   await MySQLDataSource.initialize();
   console.log(MySQLDataSource.isInitialized)
   return  {
       ogm: ogm,
       session: session,
       updateFingerprintCacheSession: updateFingerprintCacheSession,
       processId: process.pid,
       models: {
           Customer: ogm.model( "Customer" ),
           OrcaTestConfig: ogm.model( "OrcaTestConfig" ),
           FunctionalWebBrowserConfiguration: ogm.model(
"FunctionalWebBrowserConfiguration" ),
           MobileDeviceConfiguration: ogm.model( "MobileDeviceConfiguration"
),
           MobileBaseConfig: ogm.model( "MobileBaseConfig" ),
           MobileConfig: ogm.model( "MobileConfig" ),
           SupportedSdkVersion: ogm.model( "SupportedSdkVersion" ),
           SlackNotifConfig: ogm.model( "SlackNotifConfig" ),
           TestDetail: ogm.model( "TestDetail" ),
           DnsSpoof: ogm.model( "DnsSpoof" ),
           Artifact: ogm.model( "Artifact" ),
           Endpoint: ogm.model( "Endpoint" ),
           SiteAnalysisEndpoint: ogm.model( "SiteAnalysisEndpoint" ),
           HeaderMapEntries: ogm.model( "HeaderMapEntries" ),
           HeaderMap: ogm.model( "HeaderMap" ),
           OrcaCustomer: ogm.model( "OrcaCustomer" ),
           SlackNotifConfigMap: ogm.model( "SlackNotifConfigMap" ),
           OrcaCustomerLite: ogm.model( "OrcaCustomerLite" ),
           PolicyV0: ogm.model( "PolicyV0" ),
           PolicyV1: ogm.model( "PolicyV1" ),
           EndpointMeta: ogm.model( "EndpointMeta" ),
           OrcaTest: ogm.model( "OrcaTest" ),
           ScheduledTest: ogm.model( "ScheduledTest" ),
           Test: ogm.model( "Test" ),
           ConfigMapping: ogm.model( "ConfigMapping" ),
           WebRegressionTest: ogm.model( "WebRegressionTest" ),
           TestStatus: ogm.model( "TestStatus" ),
           FirmwareV0: ogm.model( "FirmwareV0" ),
           FirmwareV1: ogm.model( "FirmwareV1" ),
           BrowserSmokeTests: ogm.model( "BrowserSmokeTests" ),
           InfraConfig: ogm.model( "InfraConfig" ),
           TestSuite: ogm.model( "TestSuite" ),
           EncodedTest: ogm.model( "EncodedTest" ),
           JobMeta: ogm.model( "JobMeta" ),
           Status: ogm.model( "Status" ),
           Label: ogm.model( "Label" ),
           FunctionalWebTest: ogm.model( "FunctionalWebTest" ),
           FunctionalMobileTestV1: ogm.model( "FunctionalMobileTestV1" ),
           FunctionalMobileTestV2: ogm.model( "FunctionalMobileTestV2" ),
           FunctionalTestV1: ogm.model( "FunctionalTestV1" ),
           FunctionalTestV2: ogm.model( "FunctionalTestV2" ),
           GenericTest: ogm.model( "GenericTest" ),
           IntegrationTestV1: ogm.model( "IntegrationTestV1" ),
           IntegrationTestV2: ogm.model( "IntegrationTestV2" ),
           Report: ogm.model( "Report" ),
           FunctionalMobileReport: ogm.model( "FunctionalMobileReport" ),
           FunctionalReport: ogm.model( "FunctionalReport" ),
           FunctionalWebReport: ogm.model( "FunctionalWebReport" ),
           IntegrationReport: ogm.model( "IntegrationReport" ),
           MobileResult: ogm.model( "MobileResult" ),
           TestReportExplanation: ogm.model( "TestReportExplanation" ),
           Message: ogm.model( "Message" ),
           IntegrationResult: ogm.model( "IntegrationResult" ),
           Expectation: ogm.model( "Expectation" ),
           Explanation: ogm.model( "Explanation" ),
           Assertion: ogm.model( "Assertion" ),

       }
   }
}
// export type MessageRaw = Omit<Message,'__typename' | 'fingerprint' | 'id'>
// export type JobMetaRaw = Omit<JobMeta,'__typename' | 'fingerprint' | 'id'>
async function printPreprocessedMobileBaseConfig(){
   const context = await getContext();
   const mobileBaseConfigRepo =
MySQLDataSource.getRepository(Mobilebaseconfig);
   const mobileBaseConfig = await mobileBaseConfigRepo.findOneBy({
       customerName: "Target"
   })
   /*
   export type JobMeta = {
   __typename?: "JobMeta";
   id: Scalars["ID"];
   policyName?: Maybe<Scalars["String"]>;
   isScheduledTest: Scalars["Boolean"];
   fingerprint: Scalars["String"];
};
    */
   const jobMeta: JobMeta= {
       id: "",
       policyName: mobileBaseConfig.policyName,
       isScheduledTest: false, // don't know what this is
       fingerprint: null
   }
   const artifact: Artifact[] = (JSON.parse(mobileBaseConfig.artifacts) as
Omit<Artifact,'__typename' | 'fingerprint' | 'id'>[]).map((artifacts):
Artifact => {
       return {
           id:"",
           fingerprint: null,
           ...artifacts
       }
   })
   const messages: Message[] = (JSON.parse(mobileBaseConfig.messages) as
Omit<Message,'__typename' | 'fingerprint' | 'id'>[]).map((message):Message=>{
       return {
           id: "",
           fingerprint: null,
           ...message
       }
   })

   const mobileBaseConfigNode: NodeMap["MobileBaseConfig"] = {
       id: "",
       createdOn: mobileBaseConfig.createdOn,
       createdBy: mobileBaseConfig.createdBy,
       finishedOn: mobileBaseConfig.finishedOn,
       updatedOn: mobileBaseConfig.updatedOn,
       beh: mobileBaseConfig.beh,
       net: mobileBaseConfig.net,
       ti: mobileBaseConfig.ti,
       jobName: mobileBaseConfig.jobName,
       // customer: mobileBaseConfig.,
       // policy: mobileBaseConfig.,
       firmware: mobileBaseConfig.firmware,
       customerName: mobileBaseConfig.customerName,
       status: mobileBaseConfig.status,
       tiName: mobileBaseConfig.tiName,
       cluster: mobileBaseConfig.cluster,
       merged: mobileBaseConfig.merged,
       pcUrl: mobileBaseConfig.pcUrl,
       // jobConfig: mobileBaseConfig.jobConfig,
       result: mobileBaseConfig.result,
       messages: messages,
       artifacts: artifact,
       counter: Number(mobileBaseConfig.counter),
       pegasusGcrUrl: mobileBaseConfig.pegasusGcrUrl,
       policyName: mobileBaseConfig.policyName,
       fingerprint: null,

   };

   const mobileBaseConfigTest = await preprocessCustomer(context,
mobileBaseConfigNode);
   console.log(mobileBaseConfigTest);
}



// async function printPreprocessedCustomer() {
//     const context = await getContext();
//     const customerRepo = MySQLDataSource.getRepository(Customer);
//     const customer = await customerRepo.findOneBy({
//         customer: "Target"
//     })
//     const customerNode: NodeMap["Customer"] = {
//         nodetype: "Customer",
//         eid: customer.eid,
//         customerName: customer.customer,
//         cluster: customer.cluster,
//         taasAvailability: customer.taaS,
//         fingerprint: null
//     };
//
//     const customerAST = await preprocessCustomer(context, customerNode);
//     console.log(customerAST);
//
// }


// printPreprocessedCustomer().catch((error) => console.error(error));
printPreprocessedMobileBaseConfig().catch((error) => console.error(error));
```

/types/
index.ts

```
import { OGM } from "@neo4j/graphql-ogm";
import { Session } from "neo4j-driver";
// import { CustomerAST } from "./apiTypes/Customer";
import {
 OrcaCustomer, OrcaTestConfig,
 ArtifactCreateInput, AssertionCreateInput,
 BrowserSmokeTestsCreateInput,
 ConfigMappingCreateInput,
 CustomerCreateInput,
 DnsSpoofCreateInput,
 EncodedTestCreateInput,
 EndpointCreateInput,
 EndpointMetaCreateInput, ExpectationCreateInput, ExplanationCreateInput,
 FirmwareV0CreateInput,
 FirmwareV1CreateInput,
 FunctionalMobileReportCreateInput,
 FunctionalMobileTestV1CreateInput,
 FunctionalMobileTestV2CreateInput,
 FunctionalReportCreateInput,
 FunctionalTestV1CreateInput,
 FunctionalTestV2CreateInput,
 FunctionalWebBrowserConfigurationCreateInput,
 FunctionalWebReportCreateInput,
 FunctionalWebTestCreateInput,
 GenericTestCreateInput,
 HeaderMapCreateInput,
 HeaderMapEntriesCreateInput,
 InfraConfigCreateInput,
 IntegrationReportCreateInput,
 IntegrationResultCreateInput,
 IntegrationTestV1CreateInput,
 IntegrationTestV2CreateInput,
 JobMetaCreateInput,
 LabelCreateInput,
 MessageCreateInput,
 MobileBaseConfigCreateInput,
 MobileConfigCreateInput,
 MobileDeviceConfigurationCreateInput, MobileResultCreateInput,
 ModelMap,
 OrcaCustomerCreateInput,
 OrcaCustomerLiteCreateInput,
 OrcaTestConfigCreateInput,
 OrcaTestCreateInput,
 PolicyV0CreateInput,
 PolicyV1CreateInput,
 ReportCreateInput, Scalars,
 ScheduledTestCreateInput,
 SiteAnalysisEndpointCreateInput,
 SlackNotifConfigCreateInput,
SlackNotifConfigMapCreateInput,
StatusCreateInput,
SupportedSdkVersionCreateInput,
TestCreateInput,
TestDetailCreateInput,
TestReportExplanationCreateInput,
TestStatusCreateInput,
TestSuiteCreateInput,
WebRegressionTestCreateInput,
FunctionalWebBrowserConfiguration,
MobileDeviceConfiguration,
MobileBaseConfig,
MobileConfig,
SupportedSdkVersion,
SlackNotifConfig,
TestDetail,
DnsSpoof,
Artifact,
Endpoint,
SiteAnalysisEndpoint,
HeaderMapEntries,
HeaderMap,
SlackNotifConfigMap,
OrcaCustomerLite,
PolicyV0,
PolicyV1,
EndpointMeta,
OrcaTest,
ScheduledTest,
Test,
ConfigMapping,
WebRegressionTest,
TestStatus,
FirmwareV0,
FirmwareV1,
BrowserSmokeTests,
InfraConfig,
TestSuite,
EncodedTest,
JobMeta,
Status,
Label,
FunctionalWebTest,
FunctionalMobileTestV1,
FunctionalMobileTestV2,
FunctionalTestV1,
FunctionalTestV2,
GenericTest,
IntegrationTestV1,
 IntegrationTestV2,
 Report,
 FunctionalMobileReport,
 FunctionalReport,
 FunctionalWebReport,
 IntegrationReport,
 TestReportExplanation,
 Message,
 IntegrationResult,
 MobileResult,
 Expectation,
 Assertion,
 Explanation,
 Customer,


} from "./apiTypes";
import { Fingerprint as FingerprintDigest } from "./apiTypes/Fingerprint";

export type Context = {
 processId: number
 parentFingerprintId?: string
 ogm: OGM,
 session?: Session,
 updateFingerprintCacheSession?: Session,
 models?: Pick<ModelMap,
     "OrcaTestConfig"|
     "FunctionalWebBrowserConfiguration"|
     "MobileDeviceConfiguration"|
     "MobileBaseConfig"|
     "MobileConfig"|
     "SupportedSdkVersion"|
     "SlackNotifConfig"|
     "TestDetail"|
     "DnsSpoof"|
     "Artifact"|
     "Endpoint"|
     "SiteAnalysisEndpoint"|
     "HeaderMapEntries"|
     "HeaderMap"|
     "OrcaCustomer"|
     "SlackNotifConfigMap"|
     "OrcaCustomerLite"|
     "Customer"|
     "PolicyV0"|
     "PolicyV1"|
     "EndpointMeta"|
     "OrcaTest"|
     "ScheduledTest"|
     "Test"|
     "ConfigMapping"|
     "WebRegressionTest"|
     "TestStatus"|
     "FirmwareV0"|
     "FirmwareV1"|
     "BrowserSmokeTests"|
     "InfraConfig"|
     "TestSuite"|
     "EncodedTest"|
     "JobMeta"|
     "Status"|
     "Label"|
     "FunctionalWebTest"|
     "FunctionalMobileTestV1"|
     "FunctionalMobileTestV2"|
     "FunctionalTestV1"|
     "FunctionalTestV2"|
     "GenericTest"|
     "IntegrationTestV1"|
     "IntegrationTestV2"|
     "Report"|
     "FunctionalMobileReport"|
     "FunctionalReport"|
     "FunctionalWebReport"|
     "IntegrationReport"|
     "TestReportExplanation"|
     "Message"|
     "IntegrationResult"|
     "MobileResult"|
     "Expectation"|
     "Assertion"|
     "Explanation"
 >

}


export type NodeMap = {
 Customer: Customer,
 OrcaTestConfig: OrcaTestConfig,
 FunctionalWebBrowserConfiguration: FunctionalWebBrowserConfiguration,
 MobileDeviceConfiguration: MobileDeviceConfiguration,
 MobileBaseConfig: MobileBaseConfig,
 MobileConfig: MobileConfig,
 SupportedSdkVersion: SupportedSdkVersion,
 SlackNotifConfig: SlackNotifConfig,
 TestDetail: TestDetail,
 DnsSpoof: DnsSpoof,
 Artifact: Artifact,
 Endpoint: Endpoint,
 SiteAnalysisEndpoint: SiteAnalysisEndpoint,
 HeaderMapEntries: HeaderMapEntries,
 HeaderMap: HeaderMap,
 OrcaCustomer: OrcaCustomer,
 SlackNotifConfigMap: SlackNotifConfigMap
 OrcaCustomerLite: OrcaCustomerLite,
 PolicyV0: PolicyV0,
 PolicyV1: PolicyV1,
 EndpointMeta: EndpointMeta,
 OrcaTest: OrcaTest,
 ScheduledTest: ScheduledTest,
 Test: Test,
 ConfigMapping: ConfigMapping,
 WebRegressionTest: WebRegressionTest,
 TestStatus: TestStatus,
 FirmwareV0: FirmwareV0,
 FirmwareV1: FirmwareV1,
 BrowserSmokeTests: BrowserSmokeTests,
 InfraConfig: InfraConfig,
 TestSuite: TestSuite,
 EncodedTest: EncodedTest,
 JobMeta: JobMeta,
 Status: Status,
 Label: Label,
 FunctionalWebTest: FunctionalWebTest,
 FunctionalMobileTestV1: FunctionalMobileTestV1,
 FunctionalMobileTestV2: FunctionalMobileTestV2,
 FunctionalTestV1: FunctionalTestV1,
 FunctionalTestV2: FunctionalTestV2,
 GenericTest: GenericTest,
 IntegrationTestV1: IntegrationTestV1,
 IntegrationTestV2: IntegrationTestV2,
 Report: Report,
 FunctionalMobileReport: FunctionalMobileReport,
 FunctionalReport: FunctionalReport,
 FunctionalWebReport: FunctionalWebReport,
 IntegrationReport: IntegrationReport,
 TestReportExplanation: TestReportExplanation,
 Message: Message,
 IntegrationResult: IntegrationResult,
 MobileResult: MobileResult,
 Expectation: Expectation,
 Assertion: Assertion,
 Explanation: Explanation
}
```

```
export type NonUnionNodes = Omit<NodeMap,
"Firmware"|"Policy"|"FunctionalMobileTest"|"FunctionalTest"|"IntegrationTest">
export type Preprocessor = {
 [Property in keyof NonUnionNodes as `preprocess${ Property }`]: ( context:
Context, node: NodeMap[Property] ) => Promise<NodeMap[Property]>;
}
export type PreprocessorKey = keyof Preprocessor

export type CreateInput = {
 OrcaTestConfig: OrcaTestConfigCreateInput,
 FunctionalWebBrowserConfiguration:
FunctionalWebBrowserConfigurationCreateInput,
 MobileDeviceConfiguration: MobileDeviceConfigurationCreateInput,
 MobileBaseConfig: MobileBaseConfigCreateInput,
 MobileConfig: MobileConfigCreateInput,
 SupportedSdkVersion: SupportedSdkVersionCreateInput,
 SlackNotifConfig: SlackNotifConfigCreateInput,
 TestDetail: TestDetailCreateInput,
 DnsSpoof: DnsSpoofCreateInput,
 Artifacts: ArtifactCreateInput,
 Endpoint: EndpointCreateInput,
 SiteAnalysisEndpoint: SiteAnalysisEndpointCreateInput,
 HeaderMapEntries: HeaderMapEntriesCreateInput,
 HeaderMap: HeaderMapCreateInput,
 OrcaCustomer: OrcaCustomerCreateInput,
 SlackNotifConfigMap: SlackNotifConfigMapCreateInput,
 OrcaCustomerLite: OrcaCustomerLiteCreateInput,
 Customer: CustomerCreateInput,
 PolicyV0: PolicyV0CreateInput,
 PolicyV1: PolicyV1CreateInput,
 EndpointMeta: EndpointMetaCreateInput,
 OrcaTest: OrcaTestCreateInput,
 ScheduledTest: ScheduledTestCreateInput,
 Test: TestCreateInput,
 ConfigMapping: ConfigMappingCreateInput,
 WebRegressionTest: WebRegressionTestCreateInput,
 TestStatus: TestStatusCreateInput,
 FirmwareV0: FirmwareV0CreateInput,
 FirmwareV1: FirmwareV1CreateInput,
 BrowserSmokeTests: BrowserSmokeTestsCreateInput,
 InfraConfig: InfraConfigCreateInput,
 TestSuite: TestSuiteCreateInput,
 EncodedTest: EncodedTestCreateInput,
 JobMeta: JobMetaCreateInput,
 Status: StatusCreateInput,
 Label: LabelCreateInput,
 FunctionalWebTest: FunctionalWebTestCreateInput,
 FunctionalMobileTestV1: FunctionalMobileTestV1CreateInput,
 FunctionalMobileTestV2: FunctionalMobileTestV2CreateInput,
 FunctionalTestV1: FunctionalTestV1CreateInput,
 FunctionalTestV2: FunctionalTestV2CreateInput,
 GenericTest: GenericTestCreateInput,
 IntegrationTestV1: IntegrationTestV1CreateInput,
 IntegrationTestV2: IntegrationTestV2CreateInput,
 Report: ReportCreateInput,
 FunctionalMobileReport: FunctionalMobileReportCreateInput,
 FunctionalReport: FunctionalReportCreateInput,
 FunctionalWebReport: FunctionalWebReportCreateInput,
 IntegrationReport: IntegrationReportCreateInput,
 TestReportExplanation: TestReportExplanationCreateInput,
 Message: MessageCreateInput,
 IntegrationResult: IntegrationResultCreateInput,
 MobileResult: MobileResultCreateInput,
 Expectation: ExpectationCreateInput,
 Assertion: AssertionCreateInput,
 Explanation: ExplanationCreateInput
}

export type AsgCreateNode={
 [Property in keyof CreateInput as `asgCreate${ Property }`]: (context:
Context, node:NodeMap[Property])=>Promise<CreateInput[Property]>;
}

export type AsgCreateNodeKey = keyof AsgCreateNode

export type Fingerprint<T> = Omit<{
 [ Property in keyof T ] : any;
}, "fingerprint" >

export type FingerprintCacheMap =
Map<FingerprintDigest['hash'],FingerprintDigest['nodeId']>

export type VisitorReturnType = Scalars["ID"]



export type OS =
 |"WINDOWS"
 |"MACOS"
 |"LINUX"
 |"CHROMEOS"
 |"IOS"
 |"ANDROID"

export type Browser =
 |"CHROME"
 |"FIREFOX"
 |"EDGE"
```

```
export type Platform =
 |"ANDROID"
 |"IOS"

export type Protocol =
 |"TCP"
 |"UDP"
 |"SCTP"

export type OrcaService =
 |"ENABLE"
 |"DISABLE"

export type Method =
 |"GET"
 |"HEAD"
 |"POST"
 |"PUT"
 |"DELETE"
 |"CONNECT"
 |"OPTIONS"
 |"TRACE"
 |"PATCH"

export type TestType =
 |"INTEGRATION"
 |"FUNCTIONAL_MOBILE"
 |"FUNCTIONAL"
 |"FUNCTIONAL_WEB"
```

<u>util/bootstrap/</u>
generatePreprocessingFunctions.ts
Based of of code written by Maxwell at F5, but we changed it to generate our own functions for our own needs

```
import { generate, OGM } from "@neo4j/graphql-ogm";
import typeDefs from "../../apollo/typeDefs";
import * as path from 'path';
import * as ts from 'typescript';
import { Kind } from "graphql/language/kinds";
import { createPreprocessorFunction } from
"@/util/bootstrap/preprocessorCodeGen";
import {
   getGraphQLNodes,
   getName,
   printTypescriptNode,
   writeFile
} from "@/util/bootstrap/utils";

const createFunctionNodes=(ogmGeneratedCode: string):ts.VariableStatement[]=>{
   const sourceFile = ts.createSourceFile(
       "",
       ogmGeneratedCode,
       ts.ScriptTarget.ESNext,
       false,
       ts.ScriptKind.TS
   );
   const gqlNodes = getGraphQLNodes(typeDefs)
   let variableStatements: ts.VariableStatement[] = []
   sourceFile.forEachChild((node) => {
       if (
           node && (
               ts.isTypeAliasDeclaration(node) ||
               ts.isEnumDeclaration(node) ||
               ts.isClassDeclaration(node) ||
               ts.isInterfaceDeclaration(node))
       ) {
           const gqlNode = gqlNodes.find((gqlNode)=>gqlNode.name ===
node.name.text && gqlNode.kind != Kind.ENUM_TYPE_DEFINITION)
           if(gqlNode && ts.isTypeAliasDeclaration(node)){
               switch(node.type.kind){
                   case ts.SyntaxKind.TypeLiteral:
                       const typeLiteralNode = node.type as ts.TypeLiteralNode
                       if(gqlNode.name !== "AsgNode"){
                           const preprocessorFunction =
createPreprocessorFunction(typeLiteralNode, gqlNode, gqlNodes)
                           variableStatements.push(preprocessorFunction)
                       }
                       break;
                   case ts.SyntaxKind.IntersectionType:
                       const intersectionTypeNode = node.type as
ts.IntersectionTypeNode
                       intersectionTypeNode.types.forEach(member=>{
                           if(member.kind == ts.SyntaxKind.TypeLiteral){
                               const typeLiteralNode = member as
ts.TypeLiteralNode
                               const preprocessorFunction =
createPreprocessorFunction(typeLiteralNode, gqlNode, gqlNodes)
                               variableStatements.push(preprocessorFunction)
                           }
                       })
                       break;
                   default:
                       console.log(`skipping AST node generation for ->
${gqlNode.name}`)
```

```
                        break;
                    }
                }
            }
    })
    return variableStatements
}

const renderCode=(functionVariableStatements:
ts.VariableStatement[]):string=>{
    let content = "";
    let functionNames:string[] =[]
    functionVariableStatements.forEach(functionVariableStatement=>{
        if(ts.isVariableStatement(functionVariableStatement)){

content=`${content}\n${printTypescriptNode(functionVariableStatement)}`
            functionNames.push(getName(functionVariableStatement))
        }
    })
    const header = `
import { Context, Fingerprint, NodeMap, Preprocessor} from "@/types";
import { preProcessVisitor } from "@/util/tms_asg_codegen/visitor";
import crypto from 'crypto';
            `
    const footer = `export const
preprocessors:Preprocessor={\n${functionNames.join(',\n')}\n}`
    content = `${header}\n${content}\n${footer}`;
    return content;
}



const main = async() => {
    const ogm = new OGM({
        typeDefs,
        // @ts-ignore
        driver:{},
    });

    const ogmGeneratedCode = (await generate({
        ogm,
        noWrite: true,
    })) as string;

    const outputDirectory = path.join(__dirname, "../../util/tms_asg_codegen");
    const functionVariableStatements: ts.VariableStatement[] =
createFunctionNodes(ogmGeneratedCode);
    const generatedPreprocessingFunctionsText =
renderCode(functionVariableStatements);
```

```
   const newFilePath = path.join(outputDirectory, 'preprocessorFunctions.ts')
   writeFile(newFilePath, generatedPreprocessingFunctionsText);
}
main().then()
```

generateTypes.ts
Based of of code written by Maxwell at F5, but we changed it to generate our own functions for our own needs

```
import { generate, OGM} from "@neo4j/graphql-ogm";
import typeDefs from "../../apollo/typeDefs";
import * as path from 'path';
import * as ts from 'typescript';
import { Kind } from "graphql/language/kinds";
import {
   GraphQLNode,
   getGraphQLNodes,
   getName,
   printTypescriptNode,
   writeFile
} from "./utils";

export type NodeTypeGroups = Map<string, [ts.TypeAliasDeclaration |
ts.EnumDeclaration | ts.ClassDeclaration | ts.InterfaceDeclaration ]>
export type FileGroups = Map<string, string>

const getGroupName = (graphqlNodes:GraphQLNode[], typescriptNodeName: string)
=> {
   let groupName = "Misc";
   graphqlNodes.forEach((graphqlNode)=>{

if(typescriptNodeName.toUpperCase().includes(graphqlNode.name.toUpperCase())){
           groupName = graphqlNode.name;
       }
   })
   return groupName;
}

const addNodeToGroup = (node: ts.Node, groupName: string, nodeTypeGroups:
NodeTypeGroups) => {
   if (
       ts.isTypeAliasDeclaration(node) ||
       ts.isEnumDeclaration(node) ||
       ts.isClassDeclaration(node) ||
       ts.isInterfaceDeclaration(node)
   ) {
       let thisNodeTypeGroup = nodeTypeGroups.get(groupName)
       if (thisNodeTypeGroup) {
          if(!thisNodeTypeGroup.some((existingNode) =>getName(existingNode)
== getName(node))){
              nodeTypeGroups.get(groupName).push(node)
          }
      } else {
          nodeTypeGroups.set(groupName, [node]);
      }
   }
}

const buildFileGroups = (ogmGeneratedCode: string):NodeTypeGroups => {
   const sourceFile = ts.createSourceFile(
       "",
       ogmGeneratedCode,
       ts.ScriptTarget.ESNext,
       false,
       ts.ScriptKind.TS
   );
   const allGraphqlNodes = getGraphQLNodes(typeDefs);
   const nodeTypeGroups: NodeTypeGroups = new Map();
   sourceFile.forEachChild((node) => {
       if (
           node && (
               ts.isTypeAliasDeclaration(node) ||
               ts.isEnumDeclaration(node) ||
               ts.isClassDeclaration(node) ||
               ts.isInterfaceDeclaration(node))
       ) {
           const groupName = getGroupName(allGraphqlNodes, node.name.text);
           addNodeToGroup(node, groupName, nodeTypeGroups)
       }
   });
   return nodeTypeGroups
}

const renderCode = (nodeTypeGroups: NodeTypeGroups): FileGroups =>{
   let indexFileContent = '';
   let fileGroups: FileGroups = new Map();
   Array.from(nodeTypeGroups.keys()).forEach((groupName) =>{
       let typeExports: string[] = [];
       let exports: string[] = [];
       let newFileContent = "";
       let thisGroupNodes = nodeTypeGroups.get(groupName);
       thisGroupNodes.forEach(node => {
           newFileContent = `${newFileContent}\n${printTypescriptNode(node)}`
           if(ts.isTypeAliasDeclaration(node) ||
ts.isInterfaceDeclaration(node)) {
               typeExports.push(node.name.text);
           } else {
               exports.push(node.name.text);
           }
       })
       fileGroups.set(groupName, newFileContent);
       if(typeExports.length > 0) {
           indexFileContent = `${indexFileContent} export type
{\n${typeExports.join(',\n')}\n} from './${groupName}';\n`
       }
       if(exports.length > 0){
           indexFileContent = `${indexFileContent} export
{\n${exports.join(',\n')}\n} from './${groupName}';\n`
       }
   })
   fileGroups.set('index', indexFileContent)
   return fileGroups
}

const collectExports = (nodeTypeGroups: NodeTypeGroups): Map<string, string>
=>{
   const exportsMap: Map<string, string> = new Map();
   Array.from(nodeTypeGroups.keys()).forEach((groupName) => {
       let thisGroupNodes = nodeTypeGroups.get(groupName)
       thisGroupNodes.forEach(node =>{
           if (
               ts.isTypeAliasDeclaration(node) ||
               ts.isEnumDeclaration(node) ||
               ts.isClassDeclaration(node) ||
               ts.isInterfaceDeclaration(node)
           ){
               exportsMap.set(getName(node), groupName)
           }
       })
   })
   return exportsMap
}

const addImports = (fileGroups: FileGroups, nodeTypeGroups: NodeTypeGroups):
FileGroups => {
   const exportsMap = collectExports(nodeTypeGroups);
   const fileGroupsWithDependenciesImported: FileGroups = new Map();
   const visit = (node: ts.Node, dependencies:Map<string, string[]>): void =>
{
       let typeName: string|null = null;
       if(ts.isTypeReferenceNode(node)){
           const typeReferenceNode = node as ts.TypeReferenceNode
           const typeReferenceNodeIdentifier = typeReferenceNode.typeName as
ts.Identifier
           typeName = typeReferenceNodeIdentifier.text
       } else if(ts.isExpressionWithTypeArguments(node)) {
           const expressionWithTypeArguments = node as
ts.ExpressionWithTypeArguments
           const expression = expressionWithTypeArguments.expression as
ts.Expression
           typeName = expression.getText()
       }
       if(typeName){
           const exported = exportsMap.get(typeName);
           if(exported) {
               let existingDependency = dependencies.get(exported)
               if(existingDependency) {
                   if(!existingDependency.includes(typeName)){
                       existingDependency.push(typeName)
                   }
               } else {
                   dependencies.set(exported,[typeName])
               }
           }
       }
       ts.forEachChild(node, (node) =>visit(node, dependencies));
   }
   Array.from(fileGroups.keys()).filter(fileGroup=>fileGroup !==
"index").forEach((fileGroup)=>{
       const thisFileGroup = fileGroups.get(fileGroup)
       const sourceFile = ts.createSourceFile(
           '',
           thisFileGroup,
           ts.ScriptTarget.Latest,
           false
       );
       const dependencies = new Map<string, string[]>();
       ts.forEachChild(sourceFile, (node)=>visit(node, dependencies));
       if (dependencies.size > 0) {
           const importStatements =
Array.from(dependencies.keys()).filter(depFilePath=>depFilePath !=
fileGroup).map(depFilePath=>{
               return `import
{\n${dependencies.get(depFilePath).join(',\n')}\n} from './${depFilePath}';`;
           }).join('\n');
           const graphqlImportStatement = "import { DocumentNode,
SelectionSetNode } from 'graphql';"
           const newContent = graphqlImportStatement + '\n\n' +
importStatements + '\n\n' + thisFileGroup;
           fileGroupsWithDependenciesImported.set(fileGroup, newContent);
       }
   })
   fileGroupsWithDependenciesImported.set("index", fileGroups.get("index"))
   return fileGroupsWithDependenciesImported
}
```

```
const main=async()=>{
   const ogm = new OGM({
       typeDefs,
       // @ts-ignore
       driver:{},
   });

   const ogmGeneratedCode = (await generate({
       ogm,
       noWrite: true,
   })) as string;
   const outputDir = path.join(__dirname, '../../types/apiTypes');
   const nodeTypeGroups = buildFileGroups(ogmGeneratedCode);
   const fileGroups = addImports(renderCode(nodeTypeGroups), nodeTypeGroups);
   Array.from(fileGroups.keys()).forEach((fileGroup)=>{
       const filePath = path.join(outputDir, `${fileGroup}.ts`);
       const fileContent = fileGroups.get(fileGroup)
       writeFile(filePath, fileContent)
   })
}

main().then()
```

preprocessorCodeGen.ts
Based off of code written by Max at F5, but we changed it to generate our own functions for our own needs

```
import * as ts from "typescript";
import {Identifier, SyntaxKind, TypeNode, TypeReferenceNode} from
"typescript";
import {GraphQLNode} from "./utils";
import {Kind} from "graphql/language/kinds";

export const createPreprocessorFunction = (node: ts.TypeLiteralNode, gqlNode:
GraphQLNode, gqlNodes: GraphQLNode[]) => {
   const nodeName = gqlNode.name;

   const functionName = ts.factory.createIdentifier(`preprocess${nodeName}`)

   const contextParameter = ts.factory.createParameterDeclaration(
       undefined,
       undefined,
       ts.factory.createIdentifier("context"),
       undefined,

ts.factory.createTypeReferenceNode(ts.factory.createIdentifier("Context"))
   );
   const nodeParameter = ts.factory.createParameterDeclaration(
       undefined,
       undefined,
       ts.factory.createIdentifier('node'),
       undefined,
       ts.factory.createIndexedAccessTypeNode(
           ts.factory.createTypeReferenceNode(
               ts.factory.createIdentifier('NodeMap'),
               undefined
           ),
           ts.factory.createLiteralTypeNode(
               ts.factory.createStringLiteral(nodeName)
           )
       ),
   );
   const parameters = [contextParameter, nodeParameter];

   const nodePropertyVariableStatement =
createNodePropertyVariableStatements(node, gqlNodes);

   const fingerprintInputVariableStatement =
createFingerprintInputVariableStatement(node, gqlNode, gqlNodes)

   const letBindings = [...nodePropertyVariableStatement,
fingerprintInputVariableStatement]

   const returnStatement = createReturnStatement(node, gqlNode,gqlNodes)

   const functionBody = ts.factory.createBlock(
       [
           ts.factory.createVariableStatement(
               undefined,
               ts.factory.createVariableDeclarationList(
                   [
                       ts.factory.createVariableDeclaration(
                           ts.factory.createIdentifier("fingerprintHash"),
                           undefined,
                           undefined,
                           ts.factory.createCallExpression(
                               ts.factory.createPropertyAccessExpression(
                                   ts.factory.createIdentifier("crypto"),
                                   ts.factory.createIdentifier("createHash")
                               ),
                               undefined,
                               [
                                   ts.factory.createStringLiteral("md5")
                               ]
                           )
```

```
                        ),
                    ],
                    ts.NodeFlags.Let
                )
            ),
            ...letBindings,
            ts.factory.createExpressionStatement(
                ts.factory.createCallExpression(
                    ts.factory.createPropertyAccessExpression(
                        ts.factory.createIdentifier("fingerprintHash"),
                        ts.factory.createIdentifier("update")
                    ),
                    undefined,
                    [
                        ts.factory.createCallExpression(
                            ts.factory.createPropertyAccessExpression(
                                ts.factory.createIdentifier("JSON"),
                                ts.factory.createIdentifier("stringify")
                            ),
                            undefined,
                            [
                                ts.factory.createIdentifier("fingerprintInput")
                            ]
                        ),
                    ]
                )
            ),
            returnStatement
        ],
        true
    );

    return ts.factory.createVariableStatement(
        [
            ts.factory.createModifier(ts.SyntaxKind.ExportKeyword)
        ],
        ts.factory.createVariableDeclarationList(
            [
                ts.factory.createVariableDeclaration(
                    functionName,
                    undefined,
                    ts.factory.createIndexedAccessTypeNode(
                        ts.factory.createTypeReferenceNode(
                            ts.factory.createIdentifier('Preprocessor'),
                            undefined
                        ),
                        ts.factory.createLiteralTypeNode(

ts.factory.createStringLiteral(`preprocess${nodeName}`)
```

```
                        )
                    ),
                    ts.factory.createArrowFunction(
                        [

ts.factory.createModifier(ts.SyntaxKind.AsyncKeyword)
                        ],
                        undefined,
                        parameters,
                        undefined,

ts.factory.createToken(ts.SyntaxKind.EqualsGreaterThanToken),
                        functionBody
                    )
                ),
            ],
            ts.NodeFlags.Const
        )
    );
};

const createNodePropertyVariableStatements = (node: ts.TypeLiteralNode,
gqlNodes: GraphQLNode[]): ts.Statement[] => {
    return node.members.map( member => {
        if ( member.kind == ts.SyntaxKind.PropertySignature ) {
            const propertySignature = member as ts.PropertySignature;
            const propertySignatureName = propertySignature.name as Identifier;
            const propertyType = getPropertyType (propertySignature, gqlNodes);
            const isArray = propertyIsArray(propertySignature);
            if (propertyType && isArray) {
                return
createVariableStatementForArrayProperty(propertySignatureName.text,
propertyType);
            } else if (propertyType && !isArray) {
                return createTypeVariableStatement(propertySignatureName.text)
            }
        }
    }).filter(member => member !== undefined);
};

const getPropertyType = (node: ts.Node, gqlNodes: GraphQLNode[]): string|null
=> {
    if (!ts.isPropertySignature(node) && !ts.isTypeReferenceNode(node)) {
        return null;
    }

    let propertySignatureType: TypeNode =
ts.isTypeReferenceNode(node)?node:node.type;
    switch(propertySignatureType.kind) {
        case ts.SyntaxKind.LiteralType:
            return null
        case ts.SyntaxKind.TypeReference:
            const propertyTypeReferenceNode = propertySignatureType as
ts.TypeReferenceNode
            if (propertyTypeReferenceNode.typeArguments &&
propertyTypeReferenceNode.typeArguments.length > 0) {
                let nextTypeReference =
propertyTypeReferenceNode.typeArguments[0] as TypeReferenceNode
                if (nextTypeReference) {
                    return getPropertyType(nextTypeReference, gqlNodes);
                } else {
                    return null;
                }
            } else {
                const propertyTypeReferenceNodeName =
propertyTypeReferenceNode.typeName as Identifier
                const gqlNodeType = propertyTypeReferenceNodeName.text
                if (gqlNodes.some((gqlNode) => gqlNode.name === gqlNodeType &&
gqlNode.kind != Kind.ENUM_TYPE_DEFINITION)) {
                    return gqlNodeType
                } else {
                    return null
                }
            }
    }
};

const propertyIsArray = (node: ts.PropertySignature | ts.TypeReferenceNode):
boolean => {
    let propertyTypeReferenceNode: TypeNode =
ts.isTypeReferenceNode(node)?node: node.type;
    if(!propertyTypeReferenceNode) {
        return true;
    }
    if(ts.isTypeReferenceNode(propertyTypeReferenceNode)){
        const propertySignatureTypeName = propertyTypeReferenceNode.typeName as
Identifier;
        if(propertySignatureTypeName.text == "Array") {
            return true
        } else if (propertyTypeReferenceNode.typeArguments &&
propertyTypeReferenceNode.typeArguments.length > 0) {
            let nextTypeReference = propertyTypeReferenceNode.typeArguments[0]
as TypeReferenceNode;
            return propertyIsArray(nextTypeReference)
        } else {
            return false
        }
    } else {
        return false
    }
}

const createTypeVariableStatement = (attributeName: string):
ts.VariableStatement => {
    return ts.factory.createVariableStatement(
        undefined,
        ts.factory.createVariableDeclarationList(
            [
                ts.factory.createVariableDeclaration(
                    ts.factory.createIdentifier(`_${attributeName}`),
                    undefined,
                    undefined,
                    ts.factory.createAwaitExpression(
                        ts.factory.createCallExpression(
                            ts.factory.createIdentifier('preProcessVisitor'),
                            undefined,
                            [
                                ts.factory.createIdentifier("context"),
                                ts.factory.createPropertyAccessExpression(
                                    ts.factory.createIdentifier("node"),

ts.factory.createIdentifier(`${attributeName}`)
                                ),
                            ]
                        )
                    )
                )
            ],
            ts.NodeFlags.Let
        )
    )
};

const createVariableStatementForArrayProperty = (attributeName: string,
attributeType: string): ts.VariableStatement => {
    const propertyAccessExpression = ts.factory.createPropertyAccessExpression(
        ts.factory.createIdentifier("node"),
        ts.factory.createIdentifier(`${attributeName}`)
    )
    const awaitedExpression = ts.factory.createAwaitExpression(
        ts.factory.createCallExpression(
            ts.factory.createPropertyAccessExpression(
                ts.factory.createIdentifier("Promise"),
                ts.factory.createIdentifier("all")
            ),
            undefined,
            [
                ts.factory.createCallChain(
                    ts.factory.createPropertyAccessExpression(
                        propertyAccessExpression,
                        ts.factory.createIdentifier("map")
                    ),
                    undefined,
                    undefined,
                    [
                        ts.factory.createArrowFunction(

 [ts.factory.createModifier(ts.SyntaxKind.AsyncKeyword)],
                            undefined,
                            [ts.factory.createParameterDeclaration(undefined,
 undefined, `instance_of_${attributeName}`)],
                            undefined,

 ts.factory.createToken(ts.SyntaxKind.EqualsGreaterThanToken),
                            ts.factory.createAwaitExpression(
                                ts.factory.createCallExpression(

 ts.factory.createIdentifier("preProcessVisitor"),
                                    [
                                        ts.factory.createIndexedAccessTypeNode(
                                            ts.factory.createTypeReferenceNode(

 ts.factory.createIdentifier('NodeMap'),
                                                undefined
                                            ),
                                            ts.factory.createLiteralTypeNode(

 ts.factory.createStringLiteral(attributeType)
                                            )
                                        )
                                    ],
                                    [
                                        ts.factory.createIdentifier("context"),

 ts.factory.createIdentifier(`instance_of_${attributeName}`)

    const existentialCheck = ts.factory.createConditionalExpression(
        propertyAccessExpression,
        ts.factory.createToken(ts.SyntaxKind.QuestionToken),
        awaitedExpression,
        ts.factory.createToken(ts.SyntaxKind.ColonToken),
        ts.factory.createArrayLiteralExpression([], false)
    );
    return ts.factory.createVariableStatement(
        undefined,
        ts.factory.createVariableDeclarationList(
            [
                ts.factory.createVariableDeclaration(
                    ts.factory.createIdentifier(`_${attributeName}`),
                    undefined,
                    undefined,
                    existentialCheck
                ),
            ],
            ts.NodeFlags.Let
        )
    )
 };

 const createFingerprintInputVariableStatement = (node: ts.TypeLiteralNode,
 gqlNode: GraphQLNode, gqlNodes: GraphQLNode[]): ts.VariableStatement => {
    const nodeName = gqlNode.name;
    const typeReference = ts.factory.createTypeReferenceNode(
        ts.factory.createIdentifier("Fingerprint"),
        [
            ts.factory.createIndexedAccessTypeNode(
                ts.factory.createTypeReferenceNode(
                    ts.factory.createIdentifier("NodeMap"),
                    undefined
                ),
                ts.factory.createLiteralTypeNode(
                    ts.factory.createStringLiteral(nodeName)
                )
            ),
        ]
    )

    return ts.factory.createVariableStatement(
        undefined,
        ts.factory.createVariableDeclarationList(
            [
                ts.factory.createVariableDeclaration(
                    ts.factory.createIdentifier("fingerprintInput"),
                    undefined,
                    typeReference,
                    ts.factory.createObjectLiteralExpression(
                        createFingerprintInputProperties(node, gqlNodes),
                        true
                    )
                )
            ],
            ts.NodeFlags.Let
        )
    );
 };
```

```
const createFingerprintInputProperties = (node: ts.TypeLiteralNode, gqlNodes:
GraphQLNode[]): ts.PropertyAssignment[] => {
   return node.members.map(member => {
       if(member.kind == ts.SyntaxKind.PropertySignature) {
           const propertySignature = member as ts.PropertySignature;
           const propertySignatureName = propertySignature.name as Identifier;
           const nodePropertyType = getPropertyType(propertySignature,
gqlNodes);
           const isArray = propertyIsArray(propertySignature);
           if(propertySignatureName.text !== "fingerprint" &&
propertySignatureName.text !== '__typename') {
               if (nodePropertyType && isArray) {
                   return ts.factory.createPropertyAssignment(

ts.factory.createIdentifier(`${propertySignatureName.text}`),
                       createMapSortJoinExpression(propertySignatureName)
                   )
               } else if (nodePropertyType && !isArray) {
                   return ts.factory.createPropertyAssignment(

ts.factory.createIdentifier(`${propertySignatureName.text}`),
                       ts.factory.createPropertyAccessChain(

ts.factory.createIdentifier(`_${propertySignatureName.text}`),

ts.factory.createToken(ts.SyntaxKind.QuestionDotToken),
                           ts.factory.createIdentifier("fingerprint")
                       )
                   )
               } else {
                   return ts.factory.createPropertyAssignment(

ts.factory.createIdentifier(`${propertySignatureName.text}`),
                       ts.factory.createPropertyAccessExpression(
                           ts.factory.createIdentifier("node"),

ts.factory.createIdentifier(`${propertySignatureName.text}`)
                       )
                   )
               }
           }
       }
   }).filter(member => member !== undefined)
}

const createReturnStatement = (node: ts.TypeLiteralNode, gqlNode: GraphQLNode,
gqlNodes: GraphQLNode[]): ts.ReturnStatement => {
   const nodeName = gqlNode.name;
   const propertyAssignments = node.members.map(member => {
       if (member.kind == ts.SyntaxKind.PropertySignature) {
           const propertySignature = member as ts.PropertySignature;
           const propertySignatureName = propertySignature.name as Identifier;
           const nodePropertyType = getPropertyType(propertySignature,
gqlNodes);
           if(!['nodeType', 'fingerprint',
'__typename'].includes(propertySignatureName.text)) {
               if(nodePropertyType) {
                   return ts.factory.createPropertyAssignment(

ts.factory.createIdentifier(`${propertySignatureName.text}`),

ts.factory.createIdentifier(`_${propertySignatureName.text}`)
                   )
               } else {
                   return ts.factory.createPropertyAssignment(

ts.factory.createIdentifier(`${propertySignatureName.text}`),
                       ts.factory.createPropertyAccessExpression(
                           ts.factory.createIdentifier("node"),

ts.factory.createIdentifier(`${propertySignatureName.text}`)
                       )
                   )
               }
           }
       }
   }).filter(member => member !== undefined)

   return ts.factory.createReturnStatement(
       ts.factory.createObjectLiteralExpression(
           [
               ts.factory.createPropertyAssignment(
                   ts.factory.createIdentifier("nodeType"),
                   ts.factory.createStringLiteral(`${nodeName}`)
               ),
               ts.factory.createPropertyAssignment(
                   ts.factory.createIdentifier("fingerprint"),
                   ts.factory.createCallExpression(
                       ts.factory.createPropertyAccessExpression(
                           ts.factory.createIdentifier("fingerprintHash"),
                           ts.factory.createIdentifier("digest")
                       ),
                       undefined,
                       [ts.factory.createStringLiteral("hex")]
                   ),
               ),
               ...propertyAssignments
           ],
           true
       )
   );
};

const createMapSortJoinExpression = (propertySignatureName: ts.Identifier):
ts.CallExpression => {
  return ts.factory.createCallExpression(
      ts.factory.createPropertyAccessExpression(
          ts.factory.createCallExpression(
              ts.factory.createPropertyAccessExpression(
                  ts.factory.createCallExpression(
                      ts.factory.createPropertyAccessExpression(

ts.factory.createIdentifier(`_${propertySignatureName.text}`),
                          ts.factory.createIdentifier("map")
                      ),
                      undefined,
                      [
                          ts.factory.createArrowFunction(
                              undefined,
                              undefined,
                              [
                                  ts.factory.createParameterDeclaration(
                                      undefined,
                                      undefined,

ts.factory.createIdentifier(`instance_of_${propertySignatureName.text}`),
                                      undefined,
                                      undefined,
                                      undefined
                                  ),
                              ],
                              undefined,

ts.factory.createToken(ts.SyntaxKind.EqualsGreaterThanToken),
                              ts.factory.createPropertyAccessExpression(

ts.factory.createIdentifier(`instance_of_${propertySignatureName.text}`),
                                  ts.factory.createIdentifier('fingerprint')
                              )
                          ),
                      ]
                  ),
                  ts.factory.createIdentifier('sort')
              ),
              undefined,
              []
          ),
          ts.factory.createIdentifier('join')
      ),
      undefined,
      []
  );
};
```

Utils.ts
Based of of code written by Maxwell at F5, but we changed it to generate our own functions for our own needs

```
import { Kind } from "graphql/language/kinds";
import * as ts from "typescript";
import { DocumentNode, isTypeDefinitionNode } from "graphql/language";
import fs from "fs";
import { Identifier, TypeNode, TypeReferenceNode } from "typescript";

export type GraphQLNode = {
   name: string;
   kind: Kind
}

export const getGraphQLNodes = (typeDefs: DocumentNode[]): GraphQLNode[] => {
   const graphGLNodes: GraphQLNode[] = [];
   for (const doc of typeDefs) {
       const definitions = doc.definitions;
       for (const def of definitions) {
           if(isTypeDefinitionNode(def)){
               const graphQLNode: GraphQLNode = {
                   name : def.name.value,
                   kind: def.kind
               }
               graphGLNodes.push(graphQLNode);
           }
       }
   }
   return graphGLNodes;
}
export const printTypescriptNode = (node: ts.Node): string => {
   const printer = ts.createPrinter({newLine: ts.NewLineKind.LineFeed});
   return printer.printNode(ts.EmitHint.Unspecified, node,
ts.createSourceFile("", "", ts.ScriptTarget.ESNext));
}

export const writeFile = (filePath: string, content: string): void =>{
   fs.writeFileSync(filePath, content);
```

```
}

export const getName = (node: ts.TypeAliasDeclaration | ts.EnumDeclaration |
ts.ClassDeclaration | ts.InterfaceDeclaration |
ts.VariableStatement):string=>{
   if(ts.isVariableStatement(node)){
       const variableStatement = node as ts.VariableStatement
       const variableDeclarationList = variableStatement.declarationList as
ts.VariableDeclarationList
       const variableDeclaration = variableDeclarationList.declarations[0] as
ts.VariableDeclaration
       const name = variableDeclaration.name as Identifier
       return name.text
   } else {
       return node.name.text
   }
}

export const generateASTCompliantTypeLiteral=(node:
ts.TypeLiteralNode,gqlNodes:GraphQLNode[]):ts.TypeLiteralNode=>{
   let typeLiteralMembers = node.members.map(member=>{
       if(member.kind == ts.SyntaxKind.PropertySignature){
           const propertySignature = member as ts.PropertySignature;
           const propertySignatureName = propertySignature.name as
ts.Identifier
           // Rename "__typename" to "nodeName"
           if (propertySignatureName.text === '__typename') {
               return ts.factory.updatePropertySignature(
                   propertySignature,
                   propertySignature.modifiers,
                   ts.factory.createIdentifier('nodeType') as ts.Identifier,
                   undefined,
                   propertySignature.type
               );
           }
           // Remove "id" attribute
           if (propertySignatureName.text === 'id') {
               return null;
           }

           // Remove attributes ending with "Aggregate" or "Connection"
           if (propertySignatureName.text.endsWith('Aggregate') ||
propertySignatureName.text.endsWith('Connection')) {
               return null;
           }
           // Rename type references found in dexASGNodeNames with the 'AST'
prefix
           if (propertySignature.type &&
ts.isTypeReferenceNode(propertySignature.type)) {
```

```
               const typeNode = propertySignature.type;

               const renameTypeReference = (typeNode) => {
                   const typeName = (typeNode as ts.Identifier).text
                   if (gqlNodes.some((dexGraphQLNode)=>dexGraphQLNode.name ===
typeName && dexGraphQLNode.kind != Kind.ENUM_TYPE_DEFINITION)) {
                       return ts.factory.createIdentifier(`${typeName}AST`)as
ts.Identifier;
                   }
                   return typeNode;
               };

               const processGenericType = (typeNode) => {
                   if (ts.isTypeReferenceNode(typeNode)) {
                       if (typeNode.typeArguments &&
typeNode.typeArguments.length > 0) {
                           return ts.factory.createTypeReferenceNode(
                               typeNode.typeName,
                               typeNode.typeArguments.map(processGenericType)
                           )as ts.TypeReferenceNode
                       }else{
                           return
ts.factory.createTypeReferenceNode(renameTypeReference(typeNode.typeName),
undefined) as ts.TypeReferenceNode
                       }
                   }else{
                       return typeNode
                   }
               };

               const newType = processGenericType(typeNode);
               return ts.factory.updatePropertySignature(
                   propertySignature,
                   propertySignature.modifiers,
                   propertySignature.name,
                   propertySignature.questionToken,
                   newType,
               );
           }
       }
       return member;
   }).filter(member => member !== null);
   if(typeLiteralMembers.length>0){
       return ts.factory.createTypeLiteralNode(typeLiteralMembers)as
ts.TypeLiteralNode
   }else{
       throw new Error(`No members found in TypeLiteral, Could not generate a
DEX AST compliant Node.`)
   }
```

```
}
```

<u>util/tms_asg_codegen/</u>
visitor.ts
Based of of code written by Maxwell at F5, but we changed it to generate our own functions for our own needs

```
import { preprocessors } from "@/util/tms_asg_codegen/preprocessorFunctions";
import {NodeMap, PreprocessorKey, NonUnionNodes} from "@/types";
import {Context} from "@/types";



export const preProcessVisitor=async<T extends NodeMap[keyof NodeMap]>(
context:Context, astNode:any):Promise<any>=>{
   if(astNode==null){
       return null
   }
   let preprocessorKey: PreprocessorKey=`preprocess${astNode?.nodeType as
keyof NonUnionNodes}`;
   if(preprocessors.hasOwnProperty(preprocessorKey)){
       astNode = await preprocessors[preprocessorKey](context,astNode)
       return astNode as T
   }else{
       throw new Error(`Unsupported nodeType:${astNode?.nodeType}`);
   }
}
```

/util/
neo4j.ts

```
import neo4j from 'neo4j-driver'
import type { Driver } from 'neo4j-driver'

let driver: Driver

const defaultOptions = {
 uri: process.env.NEO4J_URI,
 username: process.env.NEO4J_USER,
 password: process.env.NEO4J_PASSWORD,
}

export default function getDriver() {
 const { uri, username, password } = defaultOptions
 if (!driver) {
   driver = neo4j.driver(uri, neo4j.auth.basic(username, password))
 }
 return driver
}
```

/tms-graph/
package.json (scripts)

```
{
 "private": true,
 "scripts": {
   "generatePreprocessingFunctions": "ts-node --project ./util/tsconfig.json
./util/bootstrap/generatePreprocessingFunctions.ts",
   "generateTypes": "ts-node --project ./util/tsconfig.json
./util/bootstrap/generateTypes.ts",
   "testmigration": "ts-node --project ./util/tsconfig.json
./typeORM/migrations/data_migration.ts",
   "dev": "next dev",
   "build": "next build",
   "start": "next start"
 },
 "dependencies": {
   "@apollo/client": "^3.7.3",
   "@apollo/server": "^4.3.0",
   "@as-integrations/next": "^1.2.0",
   "@neo4j/graphql": "^3.14.1",
   "@neo4j/graphql-ogm": "^3.17.2",
   "apollo-datasource": "^3.3.2",
   "deepmerge": "^4.2.2",
   "graphql": "^16.6.0",
   "mysql": "^2.18.1",
   "neo4j-driver": "^5.3.0",
   "next": "latest",
   "react": "^18.2.0",
   "react-dom": "^18.2.0",
   "reflect-metadata": "^0.1.13",
   "typescript": "^5.0.3"
 },
 "devDependencies": {
   "@types/node": "^18.16.0",
   "@types/react": "18.0.32",
   "ts-node": "^10.9.1",
   "tsconfig-paths": "^4.2.0",
   "typeorm": "^0.3.15"
 }
}
```