

# GraphQL Integration with Mulesoft

## What is GraphQL?

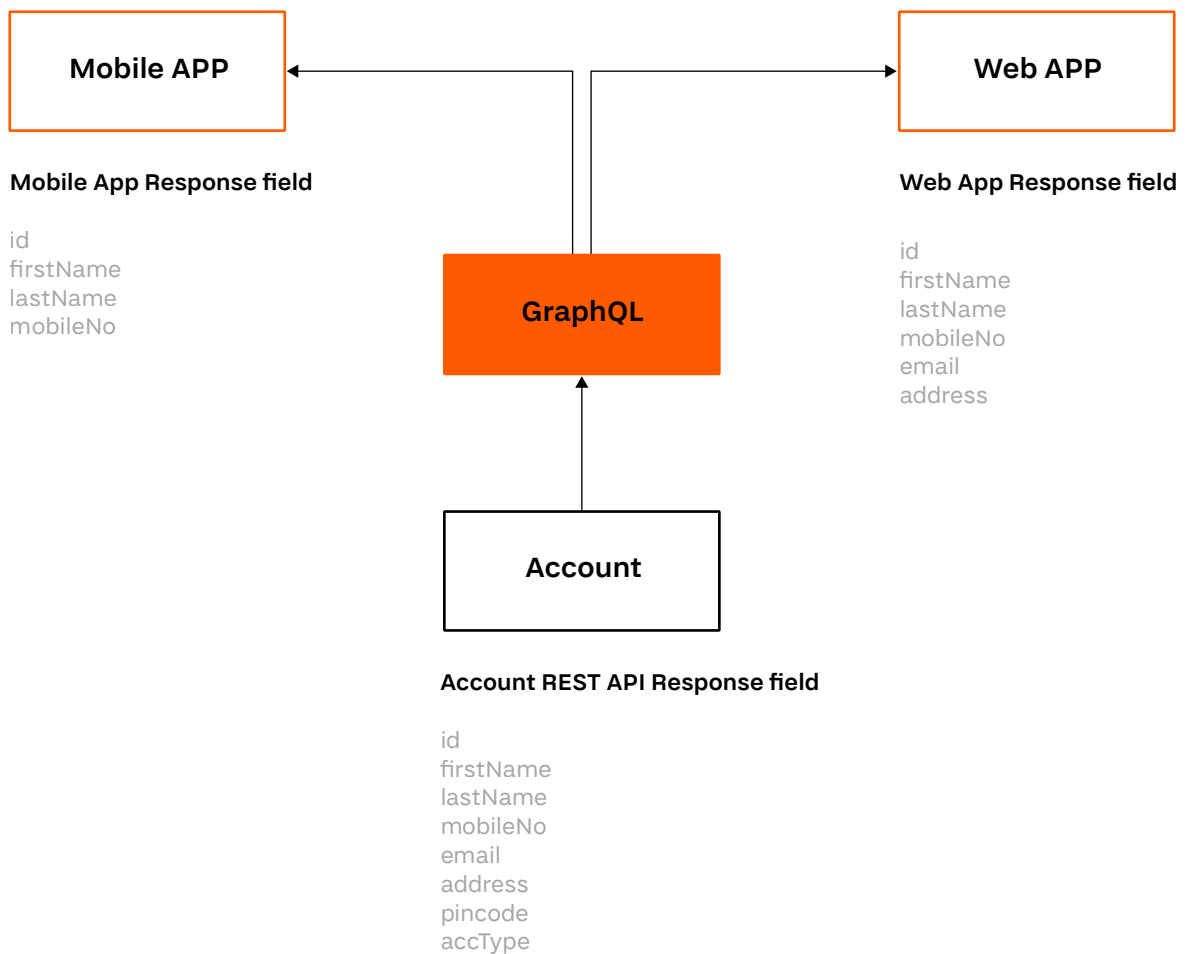
GraphQL for APIs (Application Programming Interfaces) and runtime is an open-source query language allowing clients to choose the fields they are interested in, thus cut down on the amount of data they receive when doing a query, only concentrating on what they need. GraphQL queries always return predictable results. Using GraphQL may also reduce the number of API's you need to create to meet your client's needs.

## Why GraphQL?

- Clients can handle logic as per their requirement. By using the GraphQL query they will get an expected response.
- By using GraphQL, Apps will be fast and stable because they control the data they get, not the server.
- Forget about versioning APIs. Evolve your API without versions meaning that adding more fields to an existing endpoint will not break your API.
- The frontend and backend teams can work independently. The frontend team can easily generate a schema from the backend without knowing the code. The schema generated can directly be used to create queries.

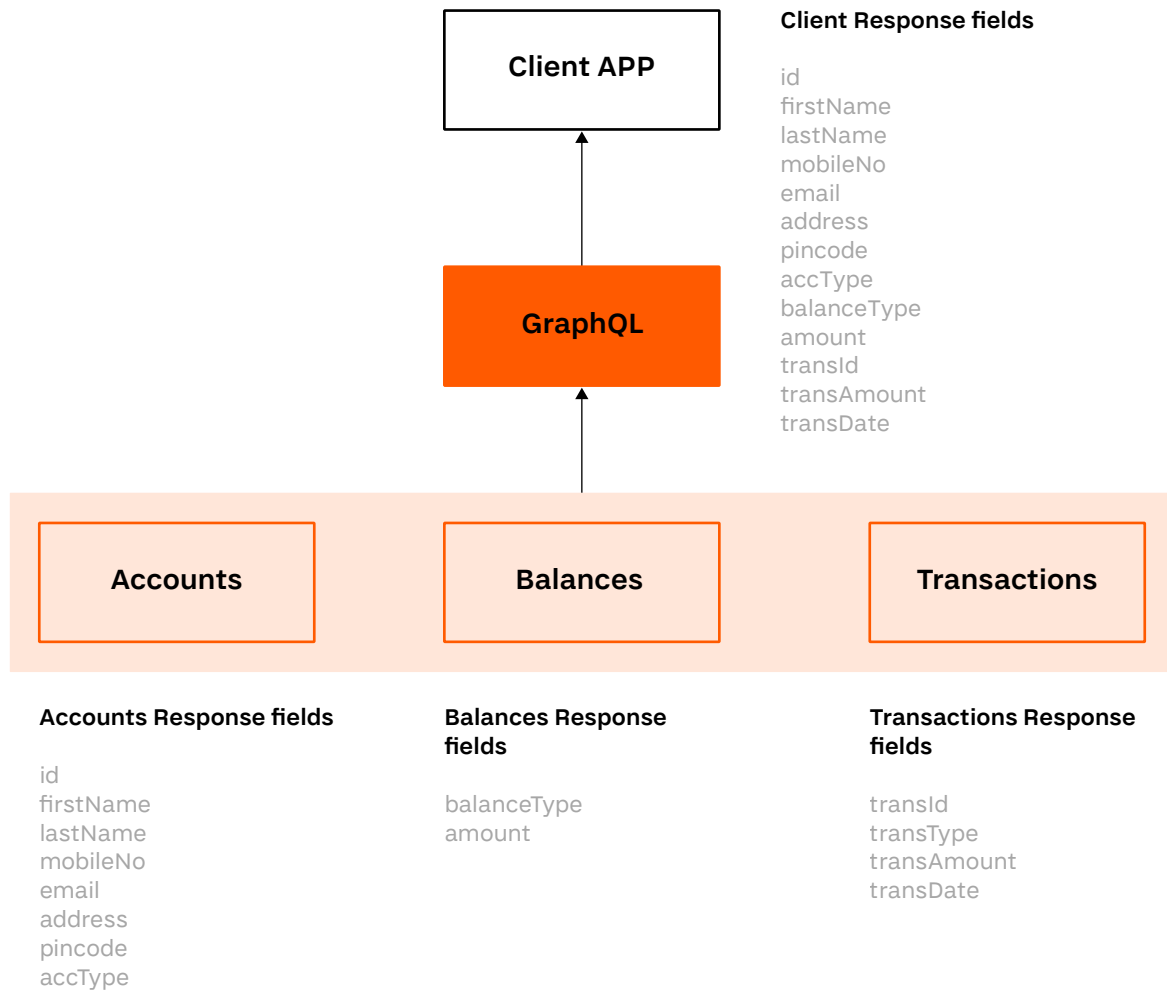
## User Case 1:

Suppose you have a REST API that returns 8 fields in response, and you have a mobile application and a web application. As per the requirement, the web application expects only 6 fields, whereas the mobile application expects only 4 fields in response. Here, according to the REST API architecture we need to create two endpoints, but with GraphQL we can achieve the above scenario with a single endpoint without any code change. Below is the illustration of User Case 1.



## User Case 2:

Suppose you must call multiple REST APIs and each API will respond with a different JSON (JavaScript Object Notation) payload. GraphQL will route multiple requests from REST API and aggregate all the responses and send back the required response to the client. Below is the illustration of User Case 2.



The above user cases can be implemented using two solutions: Anypoint Datagraph framework or GraphQL connector. We will explore each of them below.

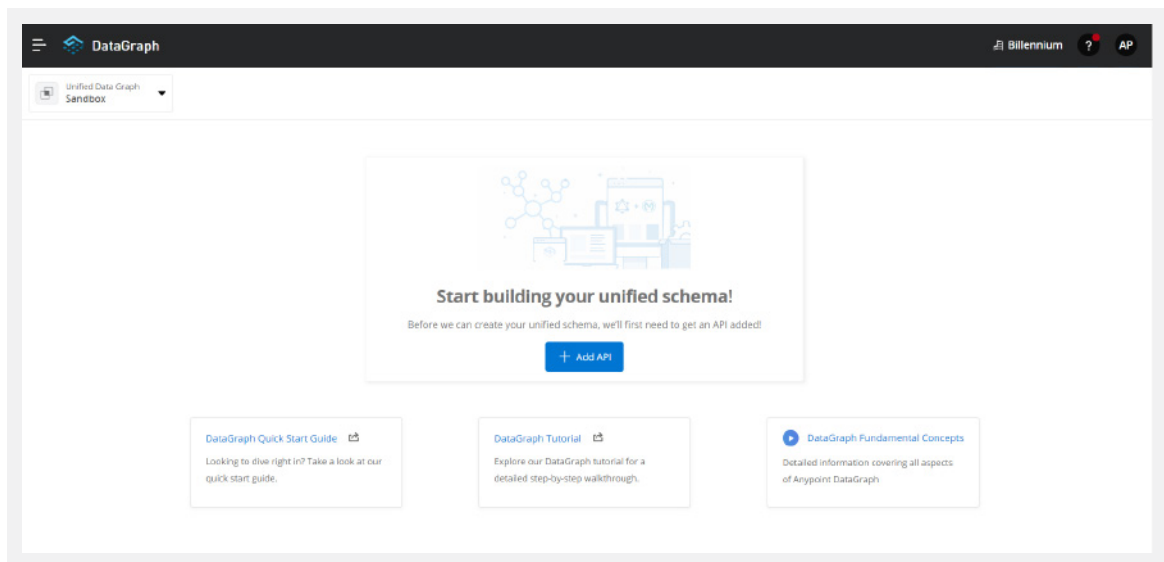
## Anypoint Platform Datagraph tool:

Anypoint Datagraph enables you to connect with these graphs into one unified schema that runs as a single SaaS (Software as a Service) GraphQL endpoint. It contains all the links and fields defined in your APIs. As a result, you can query across the underlying APIs without needing to understand all the relationships or specific capabilities that exist within them.

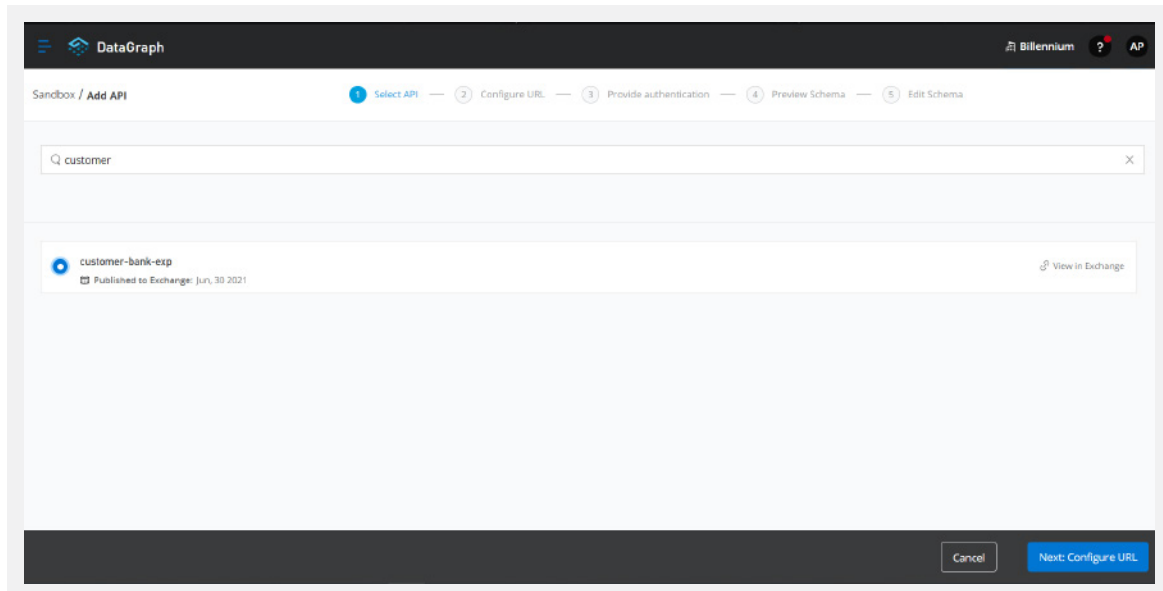
This article assumes that you are familiar with Anypoint Platform and Anypoint Studio.

Firstly, you must create an API specification in the design centre and publish RAML into exchange.

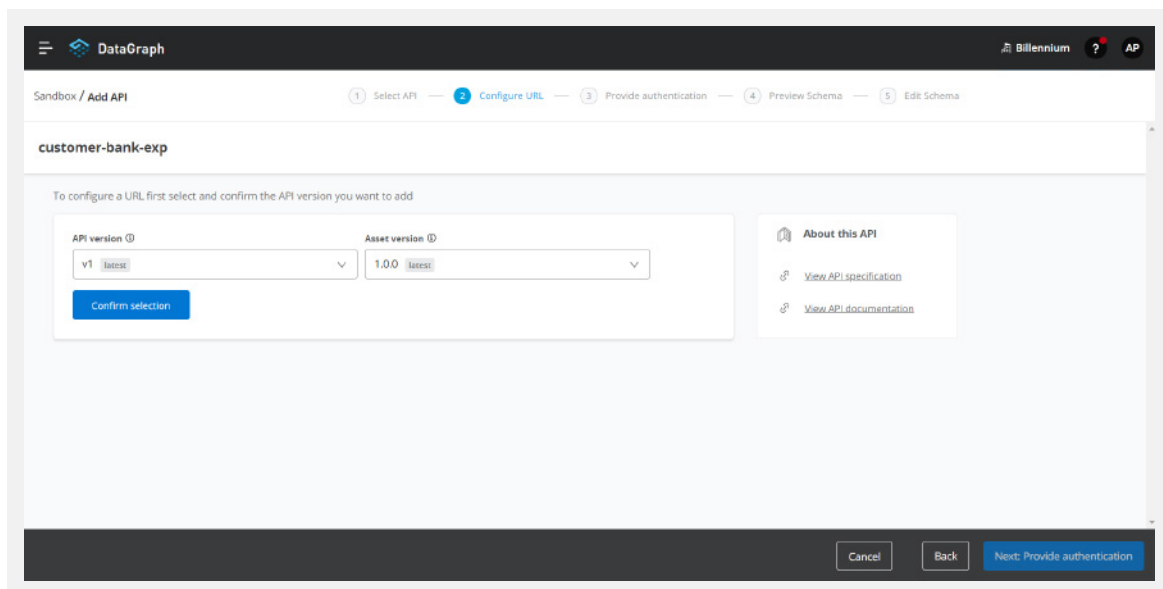
- **Step 1:**  
Go to the Anypoint Datagraph tool and select **Add API**.



- **Step 2:**  
Search your application in the search bar and after 'select application', click on **Next: Configure URL**.



- **Step 3:**  
Confirm the API and Asset version and click on **Confirm selection**.



→ **Step 4:**

After confirming the selection, provide the base URL of your application and click on **Next: Provide Authentication**. (In my case I am using a mock URL from the Design Centre)

The screenshot shows the 'Add API' configuration screen in the DataGraph interface. The breadcrumb is 'Sandbox / Add API'. The progress bar indicates five steps: 1. Select API, 2. Configure URL (current), 3. Provide authentication, 4. Preview Schema, and 5. Edit Schema. Under 'API version', 'v1 latest' is selected. Under 'Asset version', '1.0.0 latest' is selected. A green status message says 'Everything looks good'. On the right, there are links for 'View API specification' and 'View API documentation'. The main section is 'Add/Edit API URL' with the instruction 'This will be used to make requests to the API'. It has two radio buttons: 'Get an existing URL from Anypoint Platform' (selected) and 'Add a new URL'. Below the first option is a dropdown for 'API Instance in Anypoint Platform' showing 'v1:17018096'. Below that is a text field for the 'URL for the selected API instance above' showing 'https://anypoint.mulesoft.com/mocking/api/v1/links/d748339e-ea76-47de-9cde-97a3df95319b/'. An 'Edit URL' button is next to the text field. At the bottom, there are 'Cancel', 'Back', and 'Next: Provide authentication' buttons.

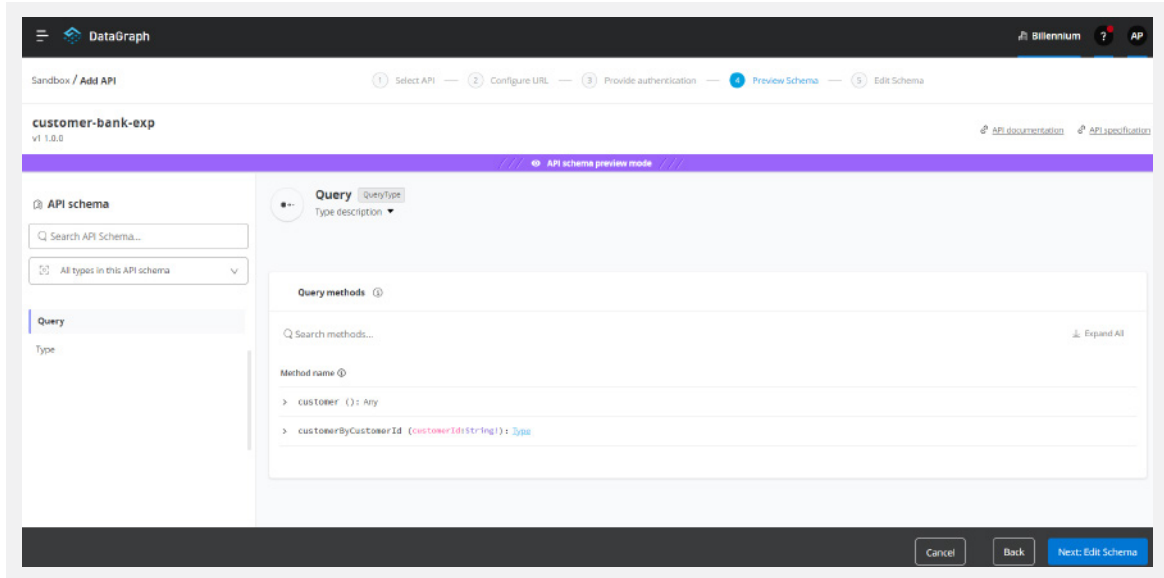
→ **Step 5:**

If you have any authentication enabled for your API, please select the authentication method accordingly and click on **Next: Preview Schema** (In my case I am using No Auth).

The screenshot shows the 'Add API' configuration screen in the DataGraph interface, now at Step 5: Provide authentication. The breadcrumb is 'Sandbox / Add API'. The progress bar indicates five steps: 1. Select API, 2. Configure URL, 3. Provide authentication (current), 4. Preview Schema, and 5. Edit Schema. The API name 'customer-bank-exp' is displayed at the top. The section is 'Provide authentication' with the instruction 'Select the authentication policy and provide credentials associated with this API's GET endpoints'. There is a dropdown for 'Select authentication policy' with 'No Auth' selected. A link 'Learn more' is next to it. On the right, there are links for 'View API specification' and 'View API documentation'. At the bottom, there are 'Cancel', 'Back', and 'Next: Preview Schema' buttons.

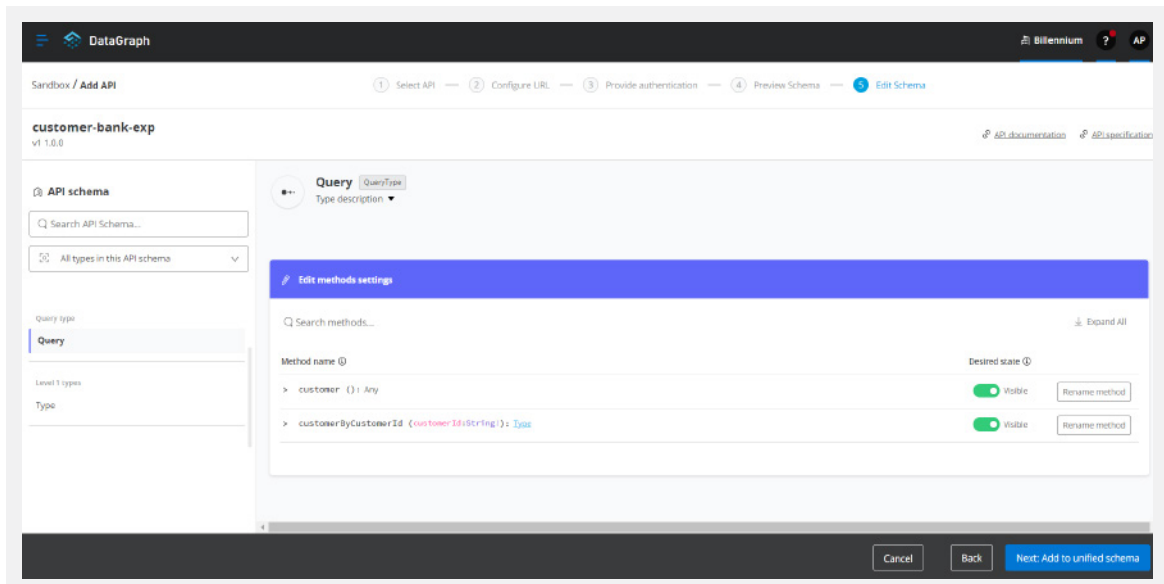
→ **Step 6:**

In the preview schema section you can see all the listed methods in your RAML and data types. Click on **Next: Edit Schema**.



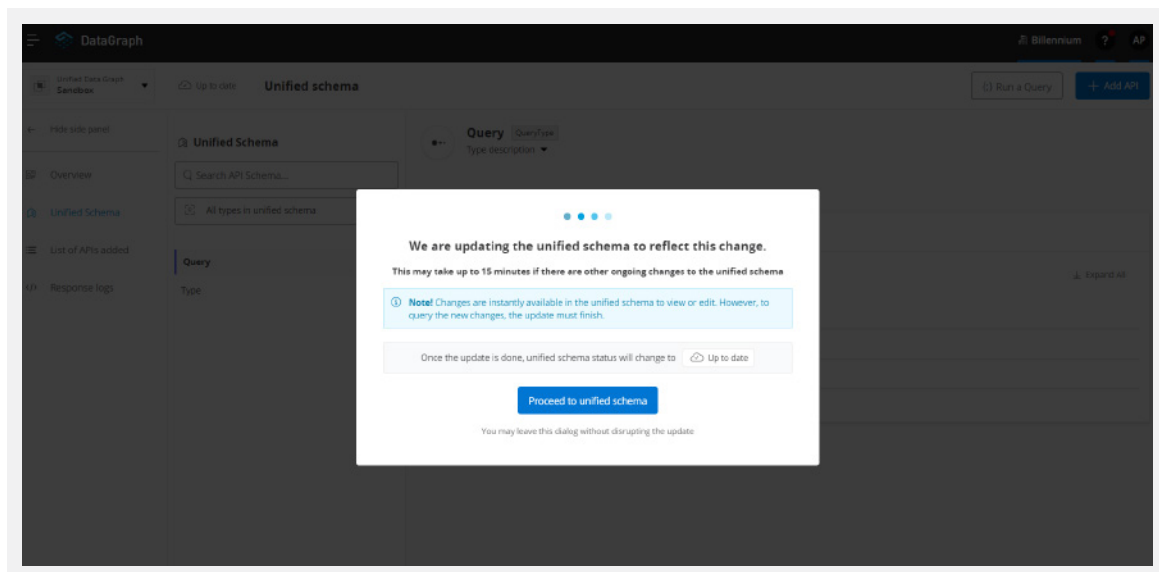
→ **Step 7:**

In the edit schema section you can hide and enable methods as per your requirement. Click on **Next: Add to unified schema**.

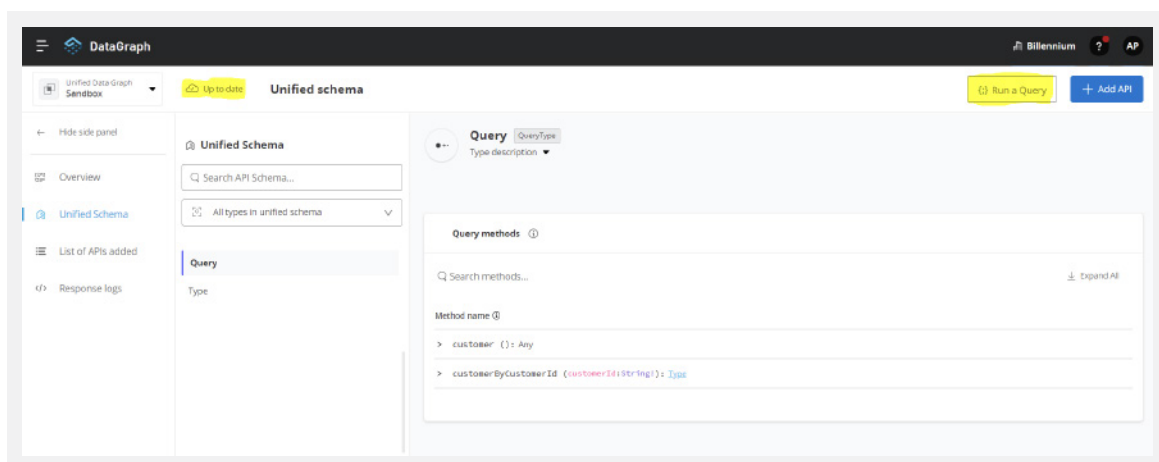




- **Step 8:**  
It may take up to 15 minutes to update the unified schema.

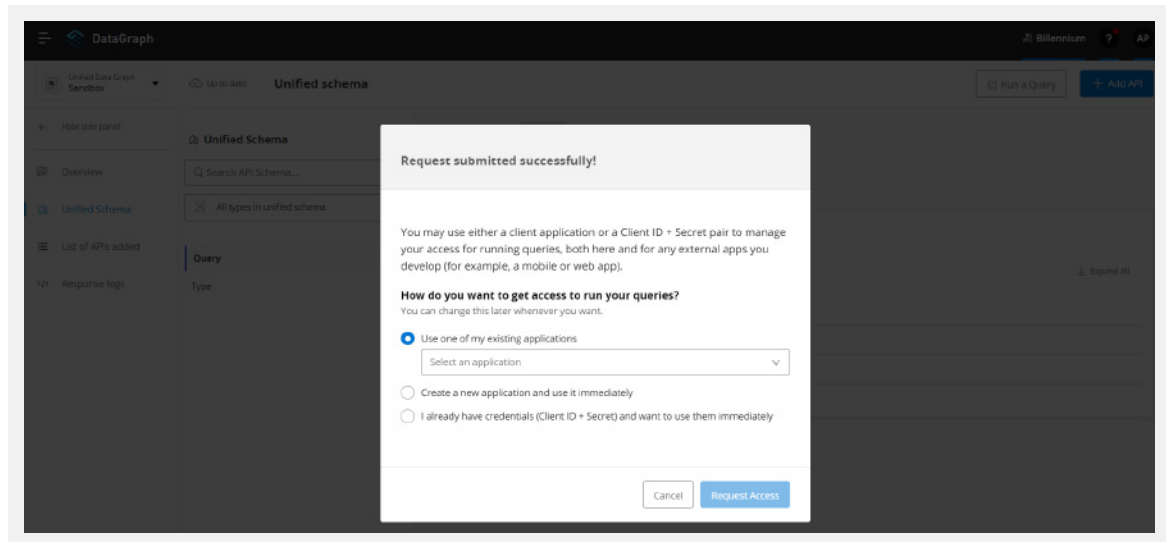


- **Step 9:**  
Once it is completed, you can see the **up to date** status as shown below and click on **Run a Query**.



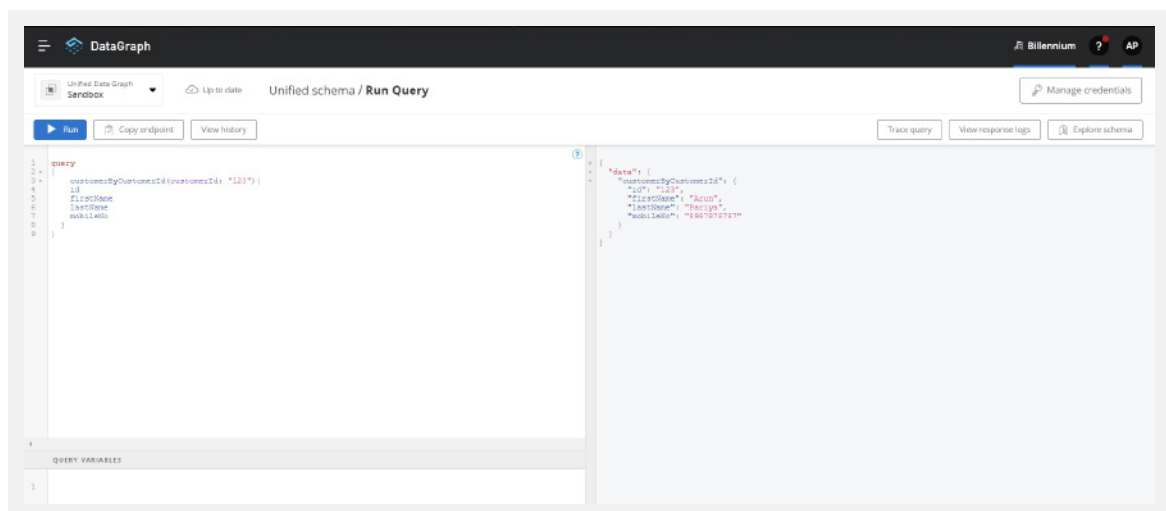
→ **Step 10:**

You can select an existing application or create a new application and click on **Request Access**.

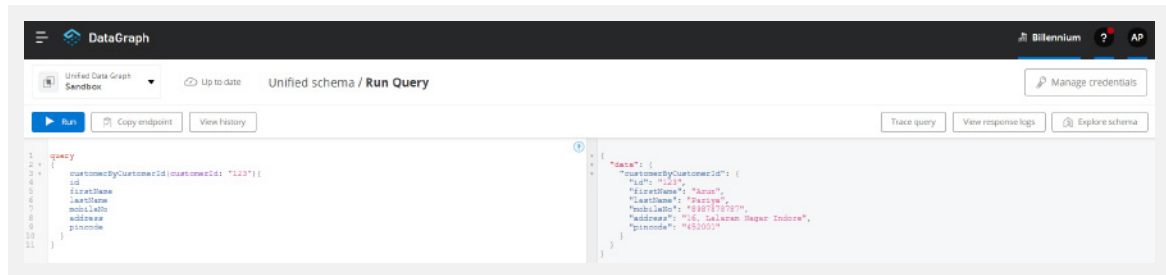


→ **Step 11:**

After application request access, your graph query console will open, and you can start to write and validate your query here.



According to your requirement, you can add more fields.



## GraphQL Implementation with Mulesoft:

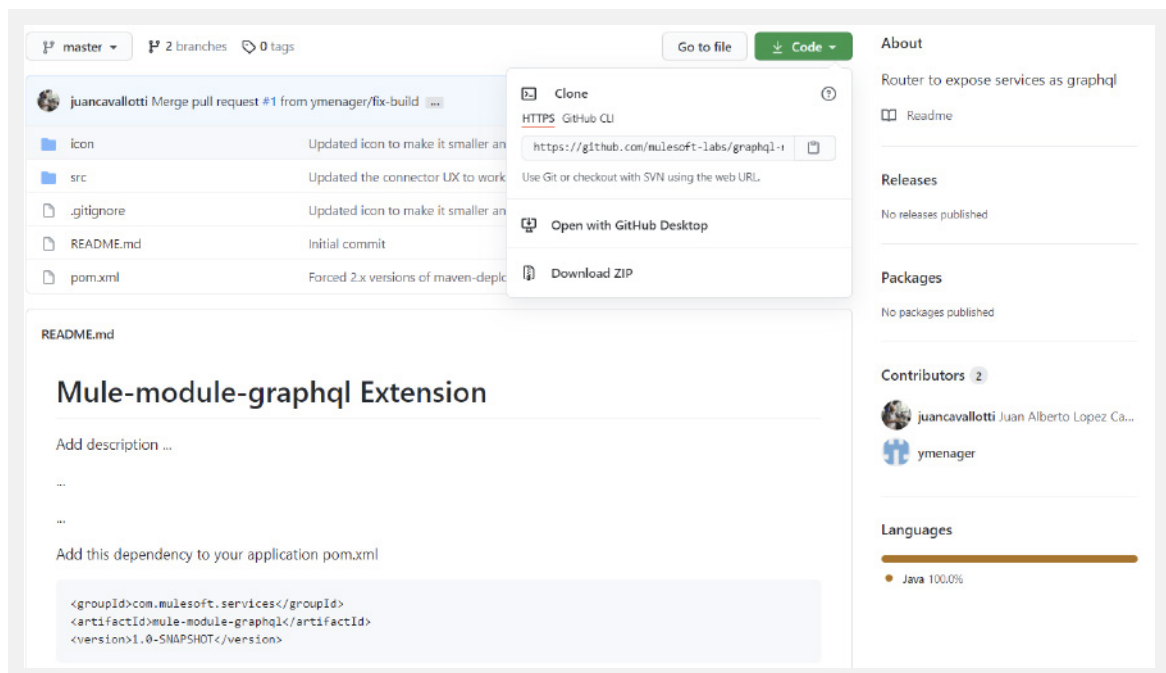
### Prerequisites:

- Anypoint Studio 7
- JDK 1.8
- Postman

Firstly, you must download the graphql router that is available on GitHub as part of open source.

Go to: <https://github.com/mulesoft-labs/graphql-router>

Copy the clone URL or download a .zip file into your local system.



After cloning or downloading the project, go to the pom.xml file and change the **mule-modules-parent** version from 1.0.0 to 1.1.3.

Add below plugin repository into your pom.xml file.

```
<pluginRepositories>
  <pluginRepository>
    <id>mulesoft-releases</id>
    <name>MuleSoft Releases Repository</name>
    <layout>default</layout>
    <url>https://repository.mulesoft.org/releases/</url>
    <snapshots>
      <enabled>>false</enabled>
    </snapshots>
  </pluginRepository>
</pluginRepositories>
```

Go to the location where the project is cloned or downloaded and then go to command prompt and run command:

```
<pluginRepositories>
```

After Building Success, let's get started...!

→ **Step 1:**

Create a sample Mule project in the Anypoint studio and add the below dependency in the pom.xml file for the GraphQL module.

```
<dependency>
  <groupId>com.mulesoft.services</groupId>
  <artifactId>mule-module-graphql</artifactId>
  <version>1.0.0-SNAPSHOT</version>
  <classifier>mule-plugin</classifier>
</dependency>
```

→ **Step 2:**

Create a sample REST API for account, balance, and transaction details.

- customer-balance-sys API

customer-balance-sys-api

Listener

Transform Message

Account balance data

Message Flow Global Elements Configuration XML

Listener x Console Problems

General

MIME Type Display Name: Listener

Redelivery Basic Settings

Responses Connector configuration: HTTP\_Listener\_config

Advanced General

Metadata Path: /api/accounts/{accountId}/balance

Notes

customer-balance-sys-api

Listener

Transform Message

Account balance data

Message Flow Global Elements Configuration XML

Account balance data x Console Problems

Output Payload

```
1 %dw 2.0
2 output application/json
3 ---
4 {
5   "balanceType": "Credit-Limit",
6   "amount": "23455"
7 }
```

- customer-account-sys API

customer-account-sys-api

Listener

Transform Message  
Customer account details

Message Flow Global Elements Configuration XML

Listener x Console Problems

General

MIME Type Display Name: Listener

Redelivery

Responses Basic Settings Connector configuration: HTTP\_Listener\_config

Advanced General

Metadata Path: /api/accounts/{accountId}

Notes

customer-account-sys-api

Listener

Transform Message  
Customer account details

Message Flow Global Elements Configuration XML

Customer account details x Console Problems

Output Payload

```
1 %dw 2.0
2 output application/json
3 ---
4 {
5   "id": "123",
6   "firstName": "Arun",
7   "lastName": "Pariya",
8   "mobileNo": "8987878787",
9   "email": "arunpariya1994@gmail.com",
10  "address": "16, Lalaram Nagar Indore",
11  "pincode": "452001",
12  "accType": "Saving"
13 }
```

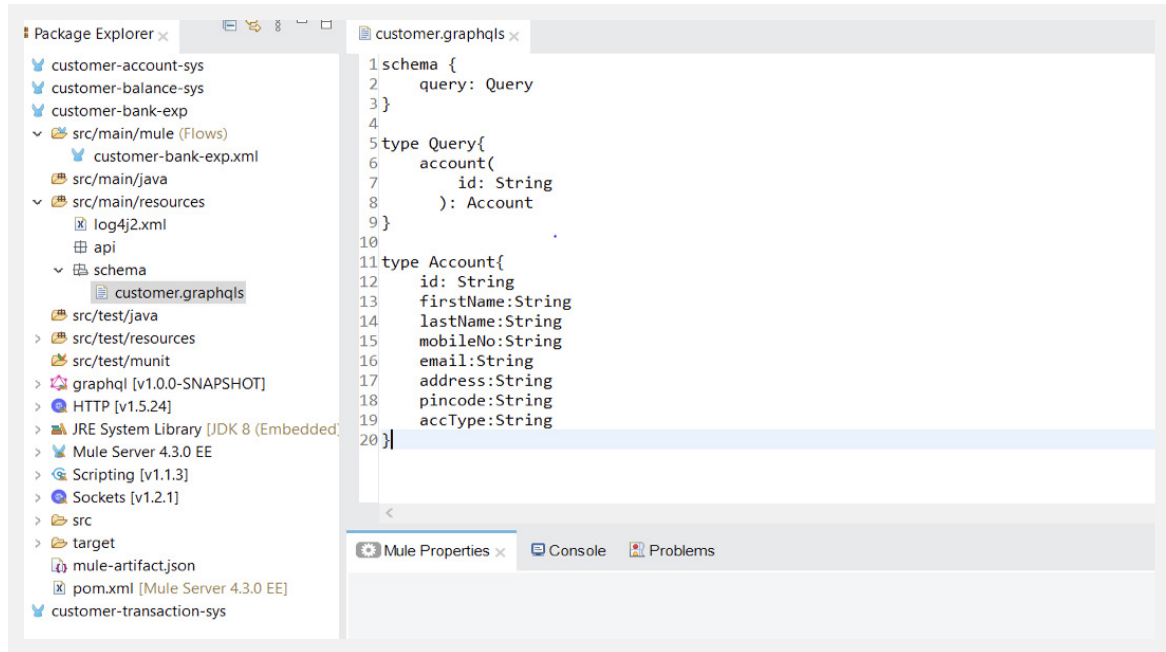
- customer-transaction-sys API

The screenshot displays the MuleSoft Anypoint Studio interface. At the top, a message flow diagram for 'customer-transaction-sysFlow' is shown, featuring a 'Listener' component connected to a 'Transform Message' component labeled 'Customer transaction details'. Below the diagram, the 'Message Flow' tab is active, showing the configuration for the selected 'Listener' component. The configuration includes a 'Display Name' of 'Listener', a 'Connector configuration' of 'HTTP\_Listener\_config', and a 'Path' of '/api/accounts/{accountId}/transactions'. The 'General' tab is selected in the left-hand pane, and the 'Console' tab is active in the bottom pane, showing a message: 'There are no errors.'

The screenshot displays the MuleSoft Anypoint Studio interface. At the top, the same message flow diagram for 'customer-transaction-sysFlow' is shown. Below the diagram, the 'Message Flow' tab is active, showing the configuration for the selected 'Customer transaction details' component. The configuration includes a 'Display Name' of 'Customer transaction details', a 'Connector configuration' of 'HTTP\_Listener\_config', and a 'Path' of '/api/accounts/{accountId}/transactions'. The 'General' tab is selected in the left-hand pane, and the 'Console' tab is active in the bottom pane, showing the output payload of the component. The output is a JSON object with the following fields: 'transId', 'transType', 'transAmount', and 'transDate'.

```
1 %dw 2.0
2 output application/json
3 ---
4 {
5   "transId": "376446",
6   "transType": "Grocery Payment",
7   "transAmount": 1345,
8   "transDate": "24-06-2021 07:22:45"
9 }
```

- **Step 3:**  
Now we will create a customer-bank-exp API.
- **Step 4:**  
Go to `/src/main/resource`, create a new folder and name it **schema**.
- **Step 5:**  
Right click on the schema Folder under `/src/main/resources` New → File.





- **Step 6:**  
Create customer-bank-exp-api flow as shown below.

The screenshot displays the MuleSoft IDE interface. At the top, a message flow diagram for 'customer-bank-exp-api' shows a 'Listener' component connected to a 'Router' component. Below the diagram, the 'Error handling' section is visible. The main panel shows the 'Listener' component configuration. The 'General' tab is selected, showing the 'Display Name' as 'Listener'. Under 'Basic Settings', the 'Connector configuration' is set to 'HTTP\_Listener\_config'. Under 'General', the 'Path' is set to '/api/graphql/customer'. The 'Console' and 'Problems' tabs are also visible at the top of the configuration panel.

customer-bank-exp-api

Listener Router

Error handling

Message Flow Global Elements Configuration XML

Listener x Console Problems

General

MIME Type

Redelivery

Responses

Advanced

Metadata

Notes

Help

Display Name: Listener

Basic Settings

Connector configuration: HTTP\_Listener\_config

General

Path: /api/graphql/customer

The screenshot displays the MuleSoft IDE interface. At the top, the same message flow diagram for 'customer-bank-exp-api' is shown. Below the diagram, the 'Error handling' section is visible. The main panel shows the 'Router' component configuration. The 'General' tab is selected, showing the 'Display Name' as 'Router'. Under 'Basic Settings', the 'Module configuration' is set to 'GraphQL\_Config'. Under 'General', the 'Payloads' are set to 'Expression'. The 'Console' and 'Problems' tabs are also visible at the top of the configuration panel.

customer-bank-exp-api

Listener Router

Error handling

Message Flow Global Elements Configuration XML

Router x Console Problems

General

Advanced

Error Mapping

Metadata

Notes

Help

Display Name: Router

Basic Settings

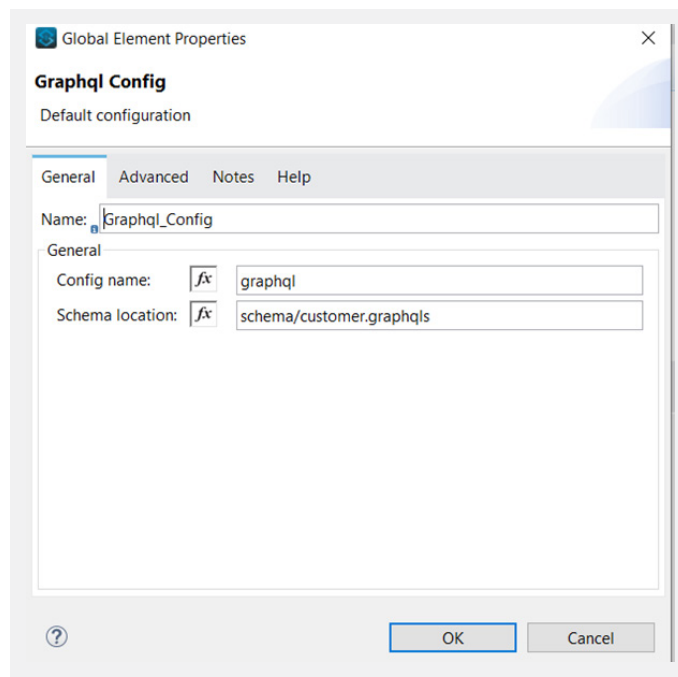
Module configuration: GraphQL\_Config

General

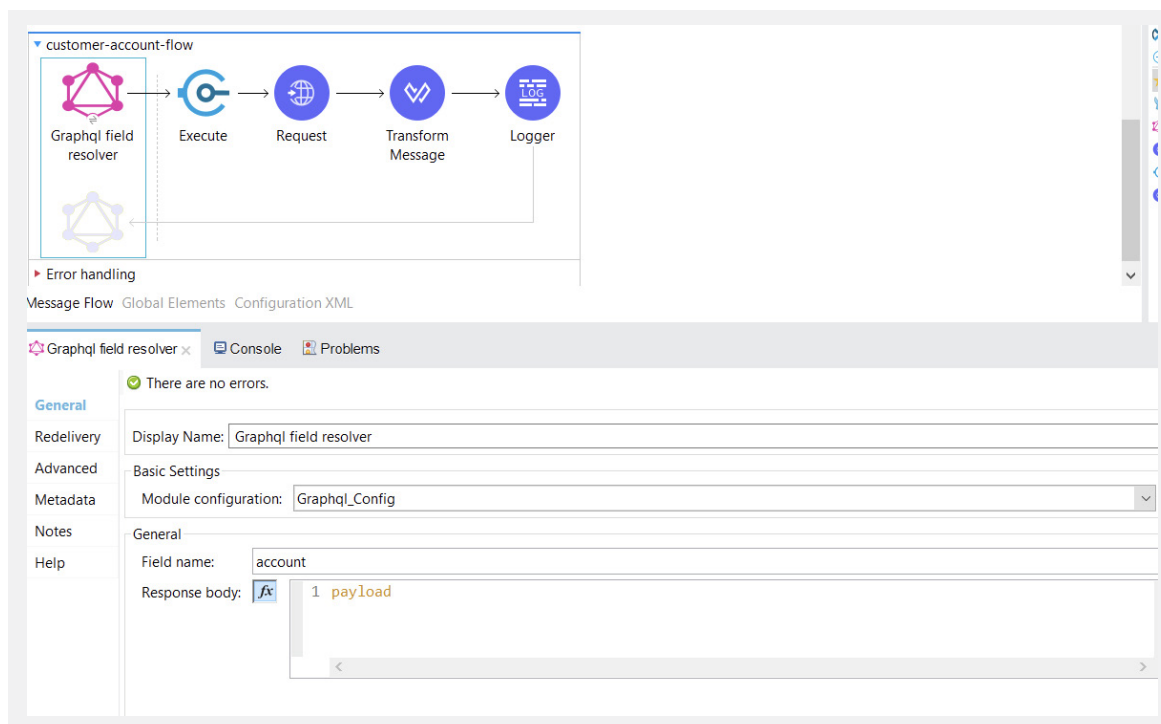
Payloads Expression

fx #[ payload

- **Step 7:**  
Configure GraphQL in global configuration as shown below.



- **Step 8:**  
Create customer account flow as show below and configure GraphQL field resolver with the field name: **account**.



→ **Step 9:**

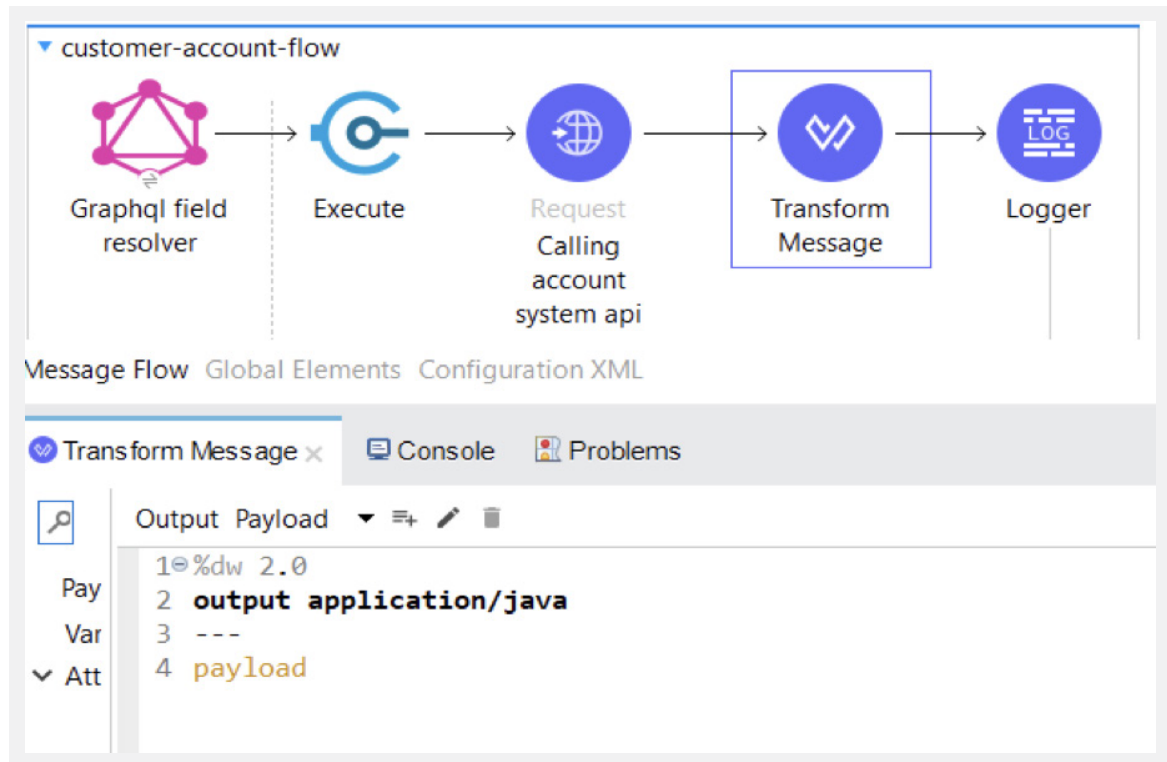
Configure the execute component. If you are not able to find the Groovy engine inside the connector then change the scripting version 1.1.3 in the pom.xml file.

The screenshot shows the MuleSoft Studio interface with a message flow named "customer-account-flow". The flow consists of five components: GraphQL field resolver, Execute, Request Calling account system api, Transform Message, and Logger. The "Execute" component is selected, and its configuration panel is open. The "General" tab is active, showing the "Display Name" as "Execute" and the "Code" as `payload = ['params': attributes.getArguments()]`. The "Engine" is set to "Groovy".

→ **Step 10:**

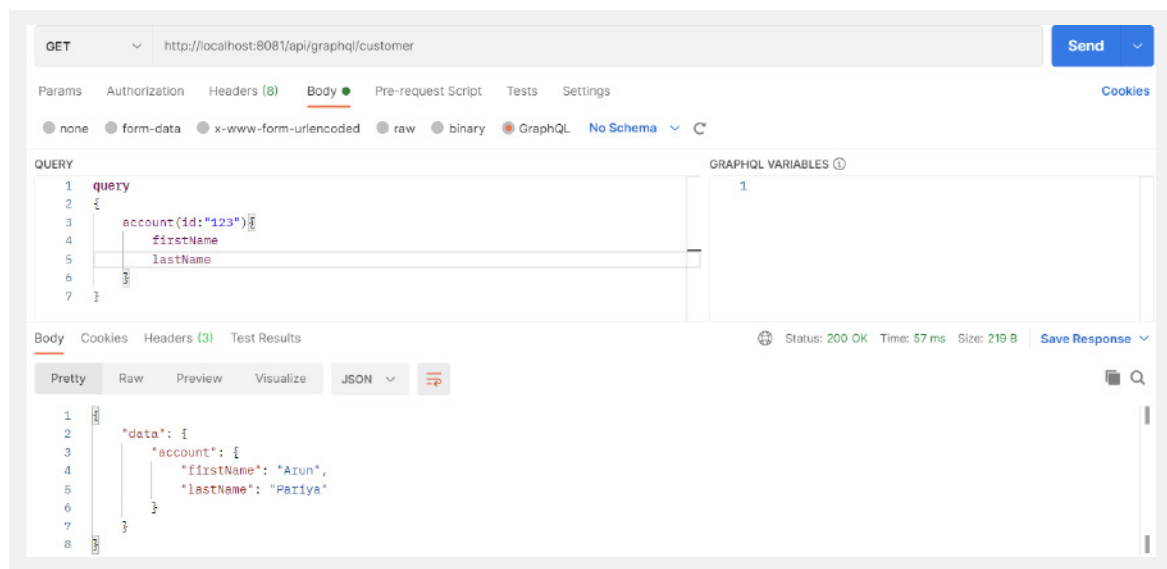
Configure the request component.

The screenshot shows the MuleSoft Studio interface with the same message flow "customer-account-flow". The "Request Calling account system api" component is selected, and its configuration panel is open. The "General" tab is active, showing the "Display Name" as "Calling account system api". The "Basic Settings" section shows the "Configuration" as "HTTP\_Request\_config\_for\_account" and the "Response" as "http://localhost:8083/api/accounts/{accountId}". The "Request" section shows the "Method" as "GET (Default)", the "Path" as "/api/accounts/{accountId}", and the "URL" as an empty field. The "Body" tab is selected, showing a table with one row: "Name" as "accountId" and "Value" as "payload.params.id".

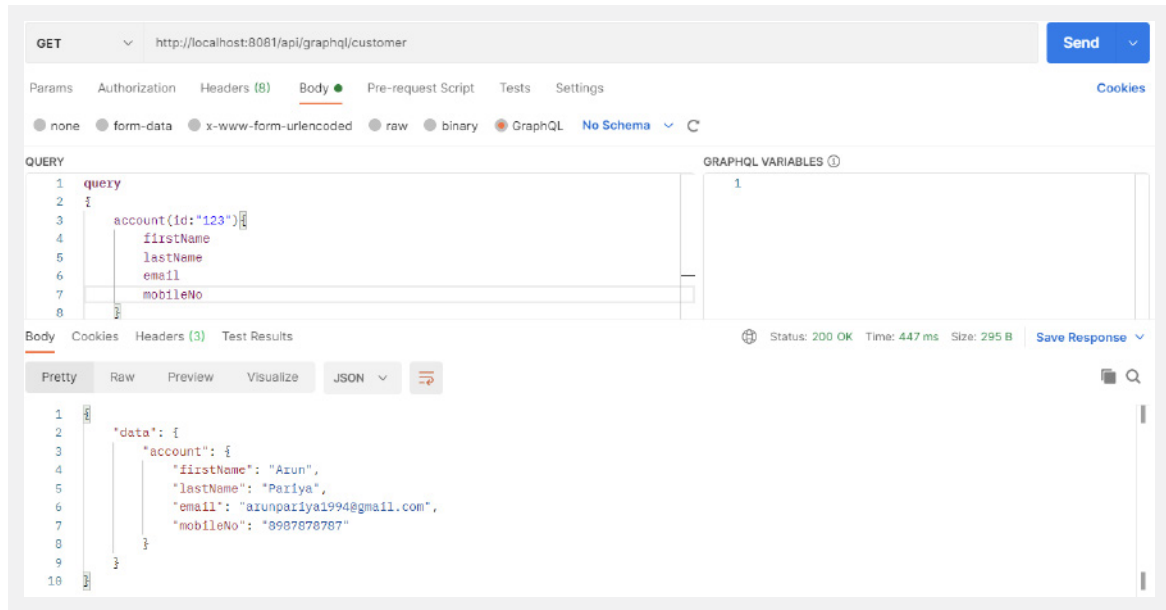


### For Use Case 1:

Go to the postman and hit graphql endpoint. Select body → GraphQL.

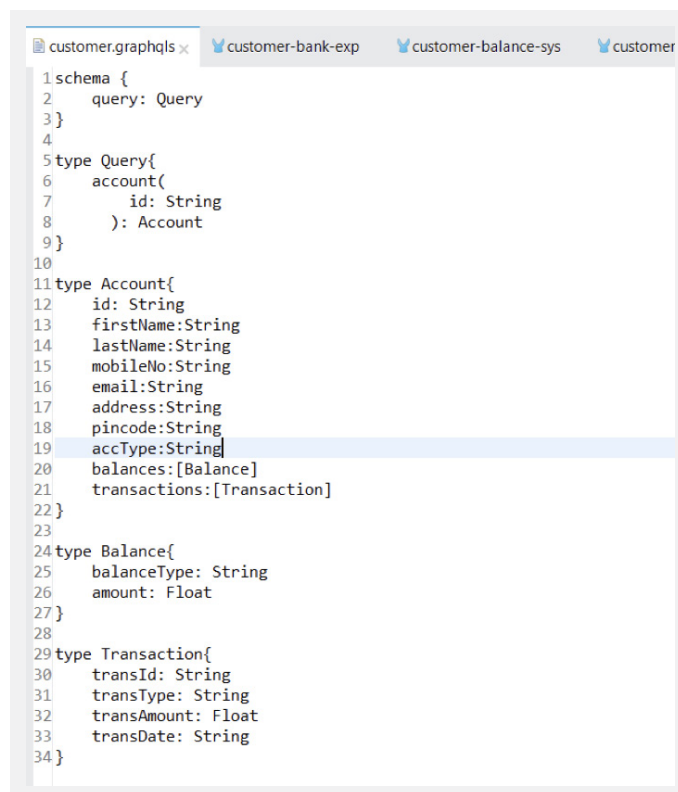


Now Change the fields according to the requirement.



## For Use Case 2:

Add the fields inside the schema under `src/main/resources/schema` folder.



- Add the balance flow as shown below.

The screenshot displays the MuleSoft Anypoint Studio interface. At the top, a message flow diagram for 'customer-balance-flow' is shown, consisting of four components in sequence: 'GraphQL field resolver', 'Execute', 'Request Calling balance system api', and 'Transform Message'. Below the diagram, the 'GraphQL field resolver' component is selected, and its configuration panel is visible. The configuration includes a 'Display Name' of 'GraphQL field resolver', a 'Module configuration' of 'GraphQL\_Config', and a 'Response body' of '1 payload'.

- Add the following script.

The screenshot displays the MuleSoft Anypoint Studio interface. At the top, the same message flow diagram for 'customer-balance-flow' is shown. Below the diagram, the 'Execute' component is selected, and its configuration panel is visible. The configuration includes a 'Display Name' of 'Execute', a 'Code' field containing the script `payload = ['params': attributes.getArguments(), 'source': attributes.getSource()]`, and an 'Engine' of 'Groovy'.

- Configure the request component as below.

The screenshot displays the MuleSoft Anypoint Studio interface. At the top, a message flow diagram for 'customer-balance-flow' is shown, consisting of four components: 'GraphQL field resolver', 'Execute', 'Request Calling balance system api', and 'Transform Message'. Below the diagram, the 'Request' component configuration panel is open. The 'General' tab is selected, showing the following settings:

- Path:** `/api/accounts/{accountId}/balance`
- URL:** (Empty)
- Body:** (Empty)
- URI Parameters:**

Name	Value
"accountId"	payload.source.id
- Send correlation id:** `-- Empty --`
- Correlation id:** (Empty)

- Add the transaction flow as show below.

The screenshot displays the MuleSoft Anypoint Studio interface. At the top, a message flow diagram for 'customer-transactions-flow' is shown, consisting of four components: 'GraphQL field resolver', 'Execute', 'Request Calling transaction system api', and 'Transform Message'. Below the diagram, the 'GraphQL field resolver' component configuration panel is open. The 'General' tab is selected, showing the following settings:

- Display Name:** GraphQL field resolver
- Basic Settings:**
  - Module configuration:** GraphQL\_Config
- General:**
  - Field name:** transactions
  - Response body:** `1 payload`

- Add the following script as shown below.

The screenshot shows a message flow named 'customer-transactions-flow' with four components: 'GraphQL field resolver', 'Execute', 'Request Calling transaction system api', and 'Transform Message'. The 'Execute' component is selected, and its configuration is shown in the right-hand pane. The 'General' tab is active, displaying the following configuration:

- Display Name: Execute
- General Code: `payload = ['params': attributes.getArguments(), 'source': attributes.getSource()]`
- Engine: `fx` Groovy
- Parameters: `fx`

The console shows a green checkmark and the message: 'There are no errors.'

- Configure the request component as shown below.

The screenshot shows the same message flow, but now the 'Request Calling transaction system api' component is selected. Its configuration is shown in the right-hand pane. The 'General' tab is active, displaying the following configuration:

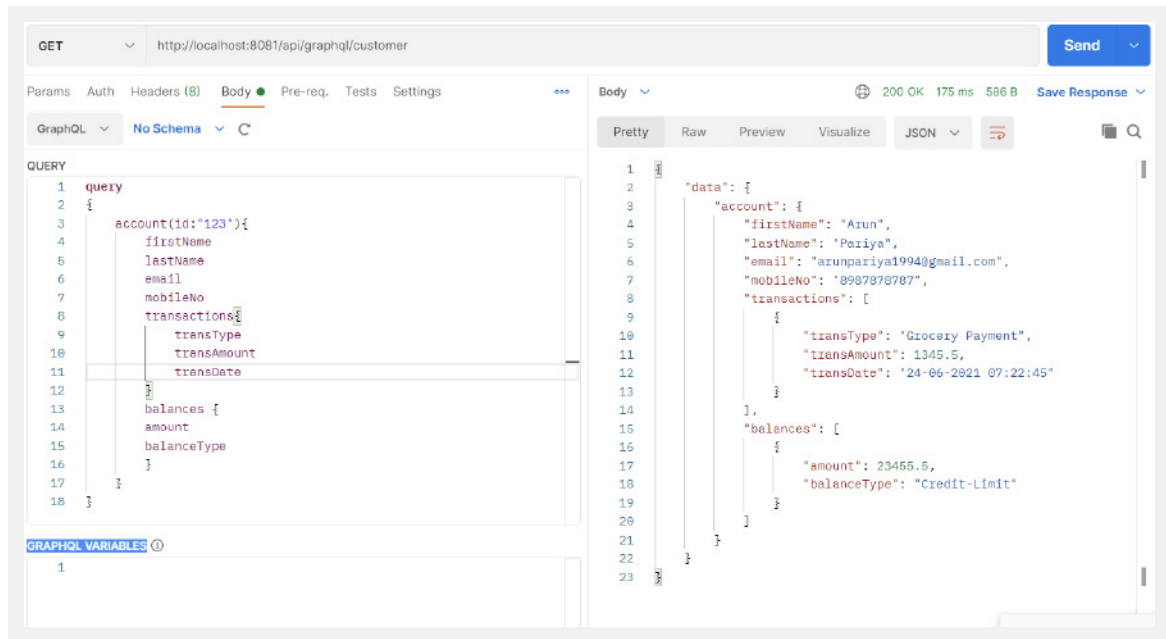
- Display Name: Calling transaction system api
- Basic Settings:
  - Configuration: HTTP\_Request\_config\_for\_transaction
  - URL: `http://localhost:8084/api/accounts/{accountId}/transactions`
- Request:
  - Method: `fx` GET (Default)
  - Path: `fx` /api/accounts/{accountId}/transactions
  - URL: `fx`
- URI Parameters:
 

Name	Value
accountId	payload.source.id

The console shows a green checkmark and the message: 'There are no errors.'



- Go to the postman and hit graphql endpoint.



This API source code is available → [here](#).

## References:

GraphQL: <https://graphql.org/>

Mulesoft: <https://anypoint.mulesoft.com/datagraph>

## Summary:

GraphQL queries are fast and stable because they control the data and get exactly what you need, nothing more, nothing less. GraphQL queries always return predictable results.