Contents

[CONTENT FRAGMENT DELIVERY VIA GRAPHQL INTERGRATION 2](#_Toc86677968)

[WHAT IS GRAPHQL? 2](#_Toc86677969)

[BENEFITS OF GRAPHQL? 2](#_Toc86677970)

[CONTENT FRAGMENT AND GRAPH QL 3](#_Toc86677971)

[DATA MODELING USING CONTENT FRAGMENTS 3](#_Toc86677972)

[ADVANCED DATA MODELING 4](#_Toc86677973)

[GRAPHQL 5](#_Toc86677974)

[EXECUTING THE GRAPHQL 6](#_Toc86677975)

[GRAPH QL ENDPOINT 10](#_Toc86677976)

[CREATING GRAPHQL ENDPOINT 10](#_Toc86677977)

[SETTINGS FOR CREATED GRAPHQL ENDPOINT 11](#_Toc86677978)

[AUTHOR AND PUBLISH ARCHITECTURE IN HEADLESS 12](#_Toc86677979)

[GRAPH-QL PERSISTED QUERY 13](#_Toc86677980)

[CREATING THE PERSISTED QUERIES (PUT REQUEST) 13](#_Toc86677981)

[EXECUTING THE PERSISTED QUERIES (GET REQUEST) 15](#_Toc86677982)

[UPDATING THE PERSISTED QUERIES 15](#_Toc86677983)

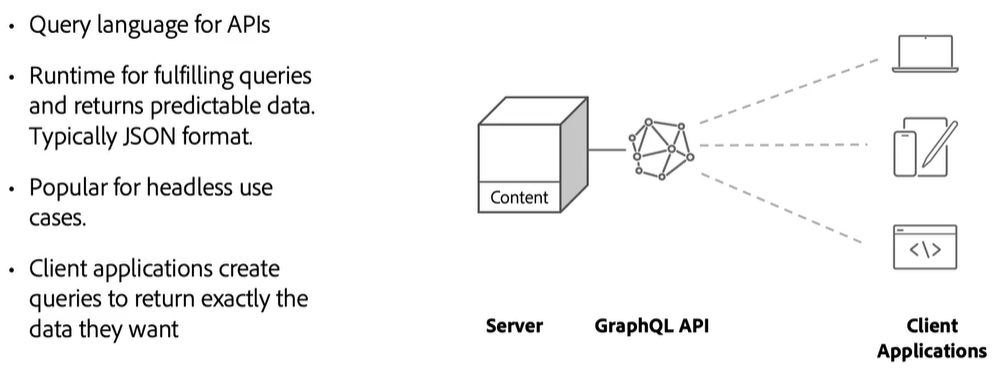
[DELETING THE PERSISTED QUERIES 16](#_Toc86677984)

[EXECUTING THE PERSISTED QUERY USING CURL 17](#_Toc86677985)

# CONTENT FRAGMENT DELIVERY VIA GRAPHQL INTERGRATION

The GraphQL API is primarily designed to deliver content fragment data to downstream application

## WHAT IS GRAPHQL?



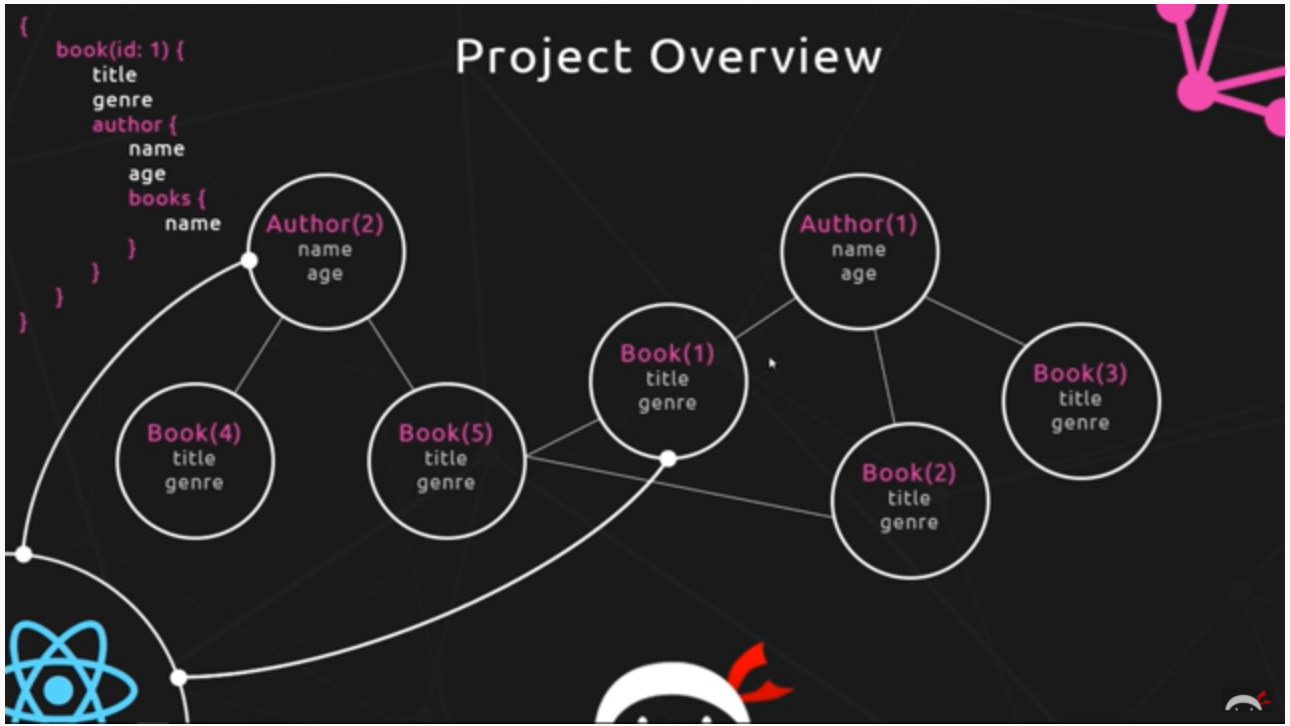
* The client applications like desktop, mobile or IOT devices can create Graph QL Queries.
* The Graph QL return the data exactly same data – what client is looking for.
* Unlike REST API – graph QL has single end point – which return the data we need.

|  |  |
| --- | --- |
|  |  |

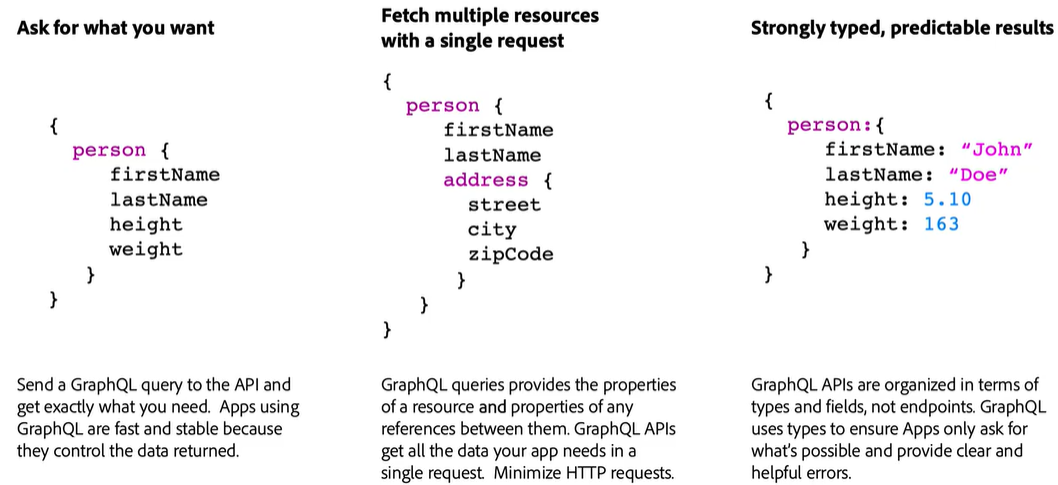
## THE GRAPH DATA STRUCTURE

* Graph QL are based on graph data structure.
* When we want to navigate to any node from application – we have a flexibility to land on different points and the further in the graph we can traverse to any related data.
* For example – lets say if we have a query as

|  |  |
| --- | --- |
| {  book(id: 1) {  title  genre  author {  name  age  books {  name  }  }  }  }} | * If we start the traversing from book (id=1). This is the entry point of the graph traversal.. We can grab the tile and genre from the node. * Next we traverse connected node of author and grab the name and age * In the next step – for the books we can traverse to next connected node of books . i.e book name - Book(2) and Book(3) |

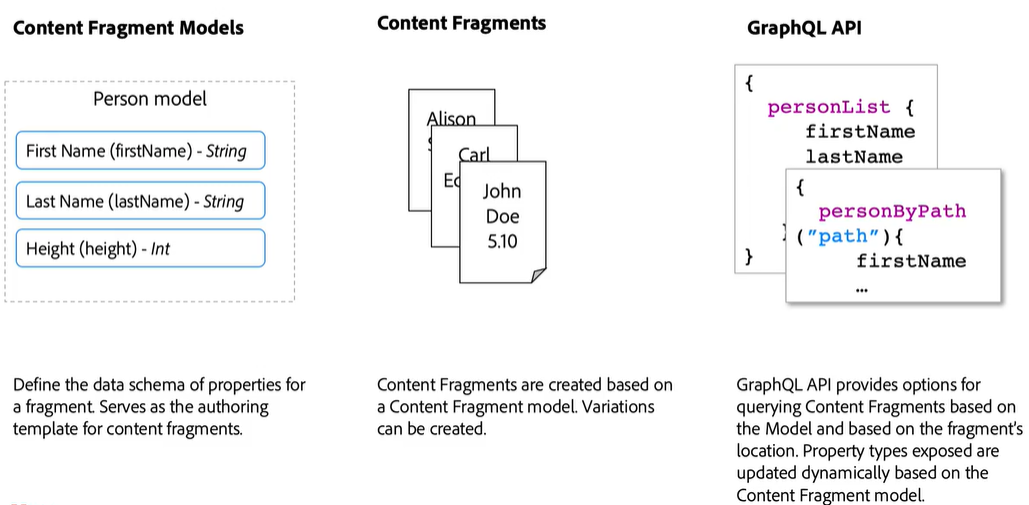


## BENEFITS OF GRAPHQL?

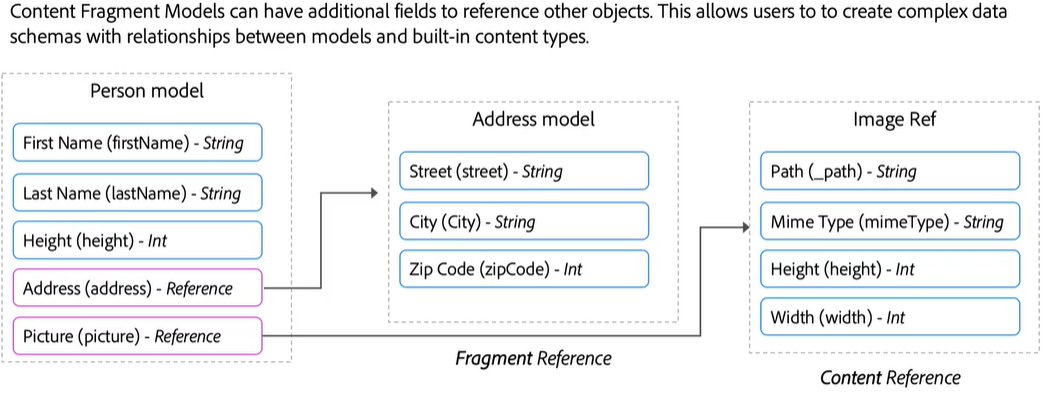


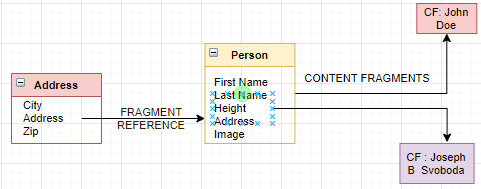
## CONTENT FRAGMENT AND GRAPH QL

AEM has implemented a set of GraphQL API to expose data about content fragment.

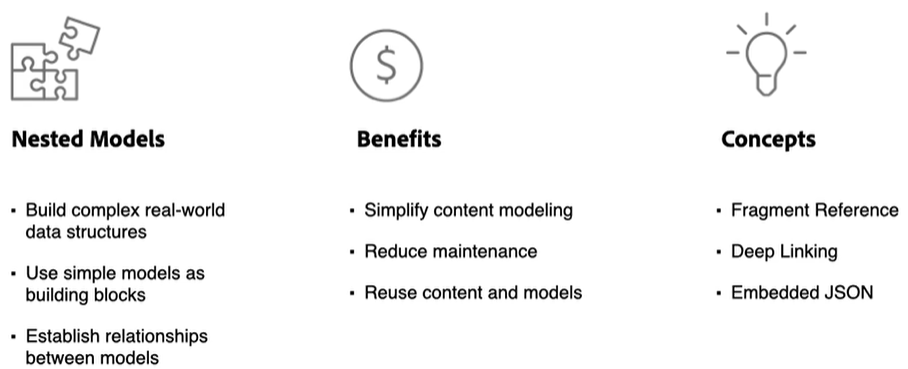


### DATA MODELING USING CONTENT FRAGMENTS





### ADVANCED DATA MODELING



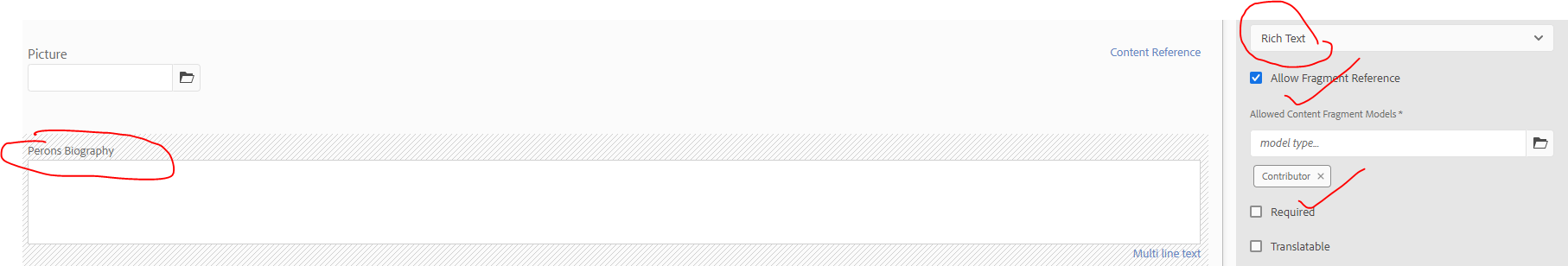
#### FRAGMENT REFERENCE

Fragment Reference field is to refer the other content fragments. Hence it helps in nesting the CFs

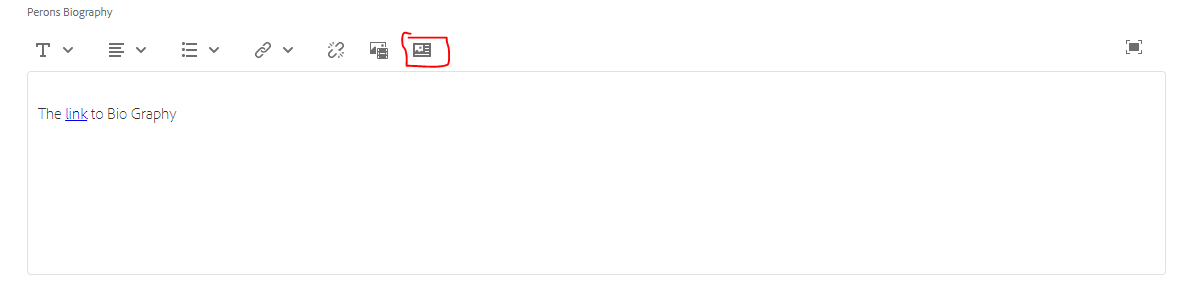
#### DEEP LINKING

* We can create a deep link to a CF in a Rich text editor.

CF MODEL

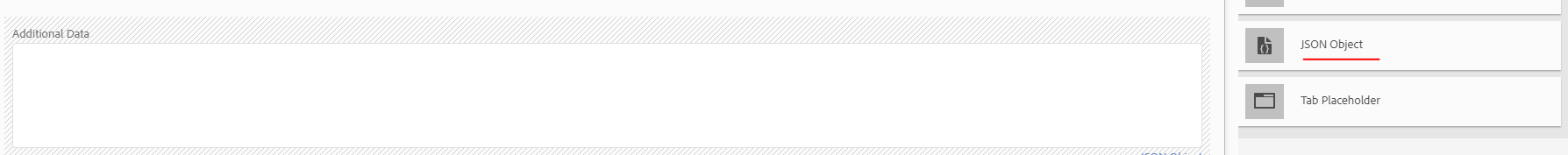


CF

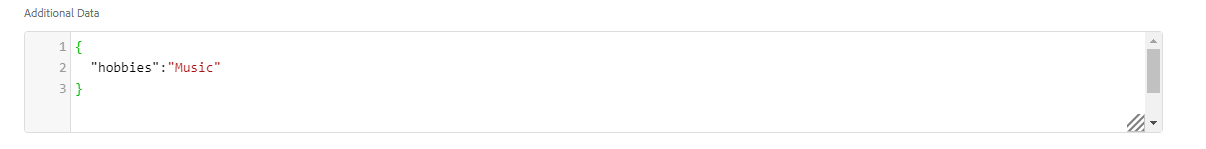


#### EMBEDDED JSON

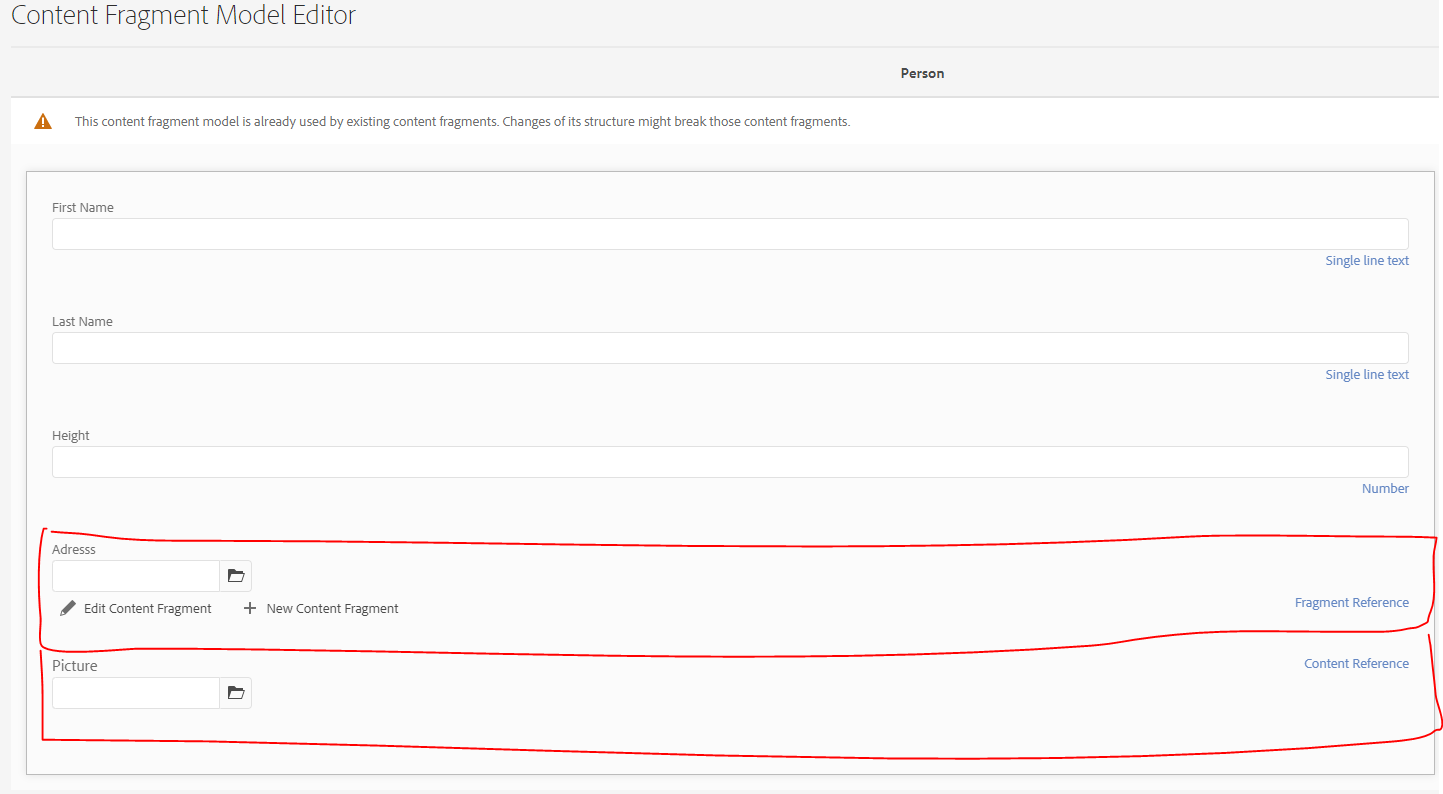
CF MODEL



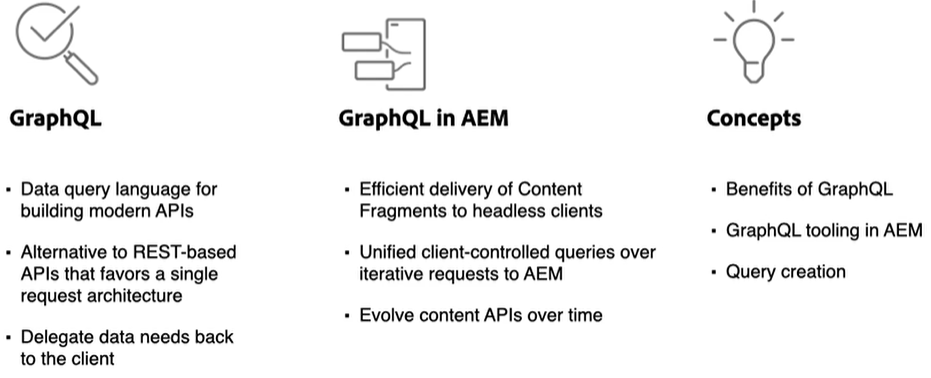
CF



#### CONTENT REFERENCE

Content Reference field is to refer the complex content types like images.  


### GRAPHQL



* AEM supports 2 query types by convention. The format of the Query type is ***modelName[ByPath|list]***

### EXECUTING THE GRAPHQL

* The GraphQL API of AEM provides a powerful query language to expose data of Content Fragments to downstream applications.
* Content Fragment Models define the data schema used by Content Fragments. Whenever a Content Fragment Model is created or updated, the schema is translated and added to the graph that makes up the GraphQL API.
* GRAPHIQL : <http://localhost:4502/content/graphiql.html>

#### QUERYING THE CONTENT FRAGMENT COLLECTION

|  |  |
| --- | --- |
| {  personList {  items {  \_path  \_variation  firstName  lastName  height  picture {  ... on ImageRef {  \_path  type  }  }  address {  street  city  zip  }  }  }  } | {  "data": {  "personList": {  "items": [  {  "\_path": "/content/dam/wknd/en/person/john-doe",  "\_variation": "master",  "firstName": "John",  "lastName": "Doe",  "height": 53,  "picture": {  "\_path": "/content/dam/wknd/en/contributors/ian\_provo.jpg",  "type": "image"  },  "address": {  "street": "2743 Lonely Oak Drive",  "city": "CALHOUN",  "zip": 42327  }  },  {  "\_path": "/content/dam/wknd/en/person/joseph-b-svoboda",  "\_variation": "master",  "firstName": "Joseph B ",  "lastName": "Svoboda",  "height": 43,  "picture": {  "\_path": "/content/dam/wknd/en/contributors/justin\_barr.jpg",  "type": "image"  },  "address": {  "street": "337 Black Oak Hollow Road",  "city": "Santa Clara",  "zip": 34  }  }  ]  }  }  } |

#### FILTERING

|  |  |
| --- | --- |
| {  personList(filter: {firstName: {\_expressions: {value: "John"}}}) {  items {  \_path  \_variation  firstName  lastName  height  picture {  ... on ImageRef {  \_path  type  }  }  address {  street  city  zip  }  }  }  } | {  "data": {  "personList": {  "items": [  {  "\_path": "/content/dam/wknd/en/person/john-doe",  "\_variation": "master",  "firstName": "John",  "lastName": "Doe",  "height": 53,  "picture": {  "\_path": "/content/dam/wknd/en/contributors/ian\_provo.jpg",  "type": "image"  },  "address": {  "street": "2743 Lonely Oak Drive",  "city": "CALHOUN",  "zip": 42327  }  }  ]  }  }  } |

#### QUERY A SINGLE FRAGMENT

* It is also possible to directly query a single Content Fragment.
* Content in AEM is stored in a hierarchical manner and the unique identifier for a fragment is based on the fragment’s path.
* If the goal is to return data about a single fragment, it is preferred to use the path and query the model directly.
* Using this syntax means that the query complexity will be very low and will generate a faster result.

|  |  |
| --- | --- |
| {  personByPath(\_path: "/content/dam/wknd/en/person/john-doe") {  item {  \_path  \_variation  firstName  lastName  height  picture {  ... on ImageRef {  \_path  type  }  }  address {  street  city  zip  }  }  }  } | {  "data": {  "personByPath": {  "item": {  "\_path": "/content/dam/wknd/en/person/john-doe",  "\_variation": "master",  "firstName": "John",  "lastName": "Doe",  "height": 53,  "picture": {  "\_path": "/content/dam/wknd/en/contributors/ian\_provo.jpg",  "type": "image"  },  "address": {  "street": "2743 Lonely Oak Drive",  "city": "CALHOUN",  "zip": 42327  }  }  }  }  } |

#### ADDING PARAMETERS TO GRAPH QL QUERIES

|  |  |
| --- | --- |
| query findPersonByPath($path: String!){  personByPath(\_path: $path) {  item {  \_path  \_variation  firstName  lastName  height  picture {  ... on ImageRef {  \_path  type  }  }  address {  street  city  zip  }  }  }  } | * “!” sign in the parameter indicate that is a required parameter * The variables are prefixed by “$” symbol   **CALLING THE FUNCTION**  {  "path": "/content/dam/wknd/en/person/john-doe"  } |



#### EXECUTING MULTIPLE GRAPH QL QUERIES IN SINGLE REQUEST

|  |  |
| --- | --- |
| {  adventureList {  items {  \_path  adventureTitle  adventurePrice  adventureTripLength  adventurePrimaryImage {  ... on ImageRef {  \_path  mimeType  width  height  }  }  }  }  personList {  items {  \_path  \_variation  firstName  lastName  height  }  }  } | * GraphQL offers the ability to specify multiple queries within a single request and therefore help reduce the round-trip times. * Here ***adventureList*** and ***personList*** are two different queries. |

#### QUERYING THE VARIATIONS

* If we query for a variation for a CF – Just in case the variation is not available, it will fall back to the master variation.

|  |  |  |
| --- | --- | --- |
| FETCHING ALL VARIATIONS:  The content fragment variations can be fetched in graplQL using variations  {  personList {  items {  \_path  \_variations  }  }  } | {  "data": {  "personList": {  "items": [  {  "\_path": "/content/dam/wknd/en/person/john-doe",  "\_variations": [  "john\_doe\_spanish"  ]  },  {  "\_path": "/content/dam/wknd/en/person/joseph-b-svoboda",  "\_variations": []  }  ]  }  }  } | |
| * Since we have the variations information available – we can be able to fetch the values of a specific variation. * If variation is not available, it will fall back to the master variation. | | |
| USING PATH  {  personByPath(  \_path: "/content/dam/wknd/en/person/john-doe"  **variation: "john\_doe\_spanish"**  ) {  item {  \_path  firstName  lastName  height  picture {  ... on ImageRef {  \_path  type  }  }  address {  street  city  zip  }  }  }  } | | {  "data": {  "personByPath": {  "item": {  "\_path": "/content/dam/wknd/en/person/john-doe",  "firstName": "John Spanish",  "lastName": "Doe Spanish",  "height": 53,  "picture": {  "\_path": "/content/dam/wknd/en/contributors/ian\_provo.jpg",  "type": "image"  },  "address": {  "street": "2743 Lonely Oak Drive",  "city": "CALHOUN",  "zip": 42327  }  }  }  }  } |
| USING LIST  {  personList(variation: "john\_doe\_spanish") {  items {  \_path  firstName  lastName  height  picture {  ... on ImageRef {  \_path  type  }  }  address {  street  city  zip  }  }  }  } | | {  "data": {  "personList": {  "items": [  {  "\_path": "/content/dam/wknd/en/person/john-doe",  "firstName": "John Spanish",  "lastName": "Doe Spanish",  "height": 53,  "picture": {  "\_path": "/content/dam/wknd/en/contributors/ian\_provo.jpg",  "type": "image"  },  "address": {  "street": "2743 Lonely Oak Drive",  "city": "CALHOUN",  "zip": 42327  }  },  {  "\_path": "/content/dam/wknd/en/person/joseph-b-svoboda",  "firstName": "Joseph B ",  "lastName": "Svoboda",  "height": 43,  "picture": {  "\_path": "/content/dam/wknd/en/contributors/justin\_barr.jpg",  "type": "image"  },  "address": {  "street": "337 Black Oak Hollow Road",  "city": "Santa Clara",  "zip": 34  }  }  ]  }  }  } |

##### FILTERING THE VARIATIONS

|  |  |
| --- | --- |
| {  personList(  variation: "**social**"  filter: {  \_variations: {  \_expressions: {  value: " **social** ",  \_operator: EQUALS  }  }  }  ) {  items {  \_path  firstName  lastName  height  picture {  ... on ImageRef {  \_path  type  }  }  address {  street  city  zip  }  }  }  } | * This will filter the exact variation of a content fragment which are names as “***social***” |

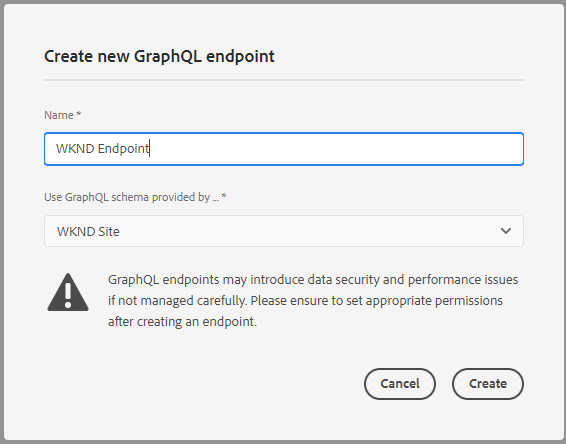
## GRAPH QL ENDPOINT

### CREATING GRAPHQL ENDPOINT

* To view all the GraphQL endpoint, navigate to Graph QL manager: Tools 🡪Assets 🡪Graph QL.
* Below is the global endpoint of Graph QL – which is will allow as to query across all the projects (system wide)



|  |  |
| --- | --- |
|  | * We can also create project specific endpoint as well. * For example: <http://localhost:4502/content/cq:graphql/wknd/endpoint.json> * The project/schema specific endpoint can only query with the project. * The global graph QL endpoint can query across all the projects. * All the graphql endpoints, CF and CFM needs to be published. |



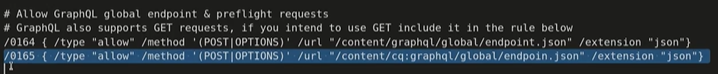


### SETTINGS FOR CREATED GRAPHQL ENDPOINT

To Allow the project specific endpoint we need to update the following

#### DISPATCHER CONFIGURATION

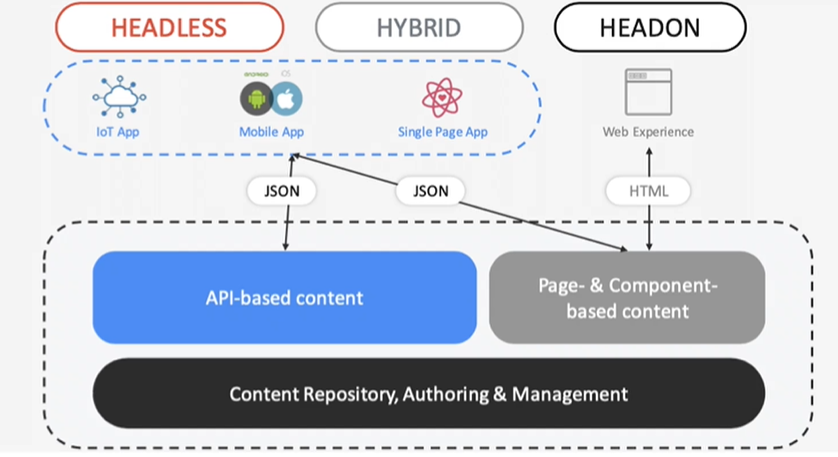
Update the dispatcher configuration to allow the endpoint.



#### SECURITY /PERMISSION CONFIGURATIONS

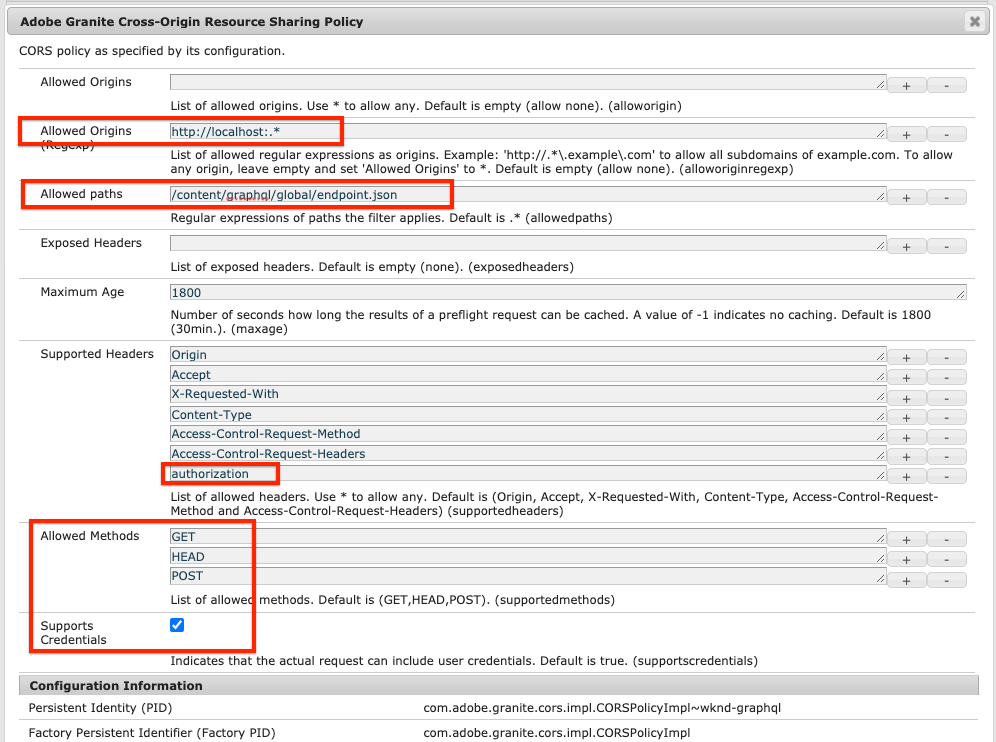
* *Navigate to : Tools 🡪 Security 🡪 Permission*

# AUTHOR AND PUBLISH ARCHITECTURE IN HEADLESS



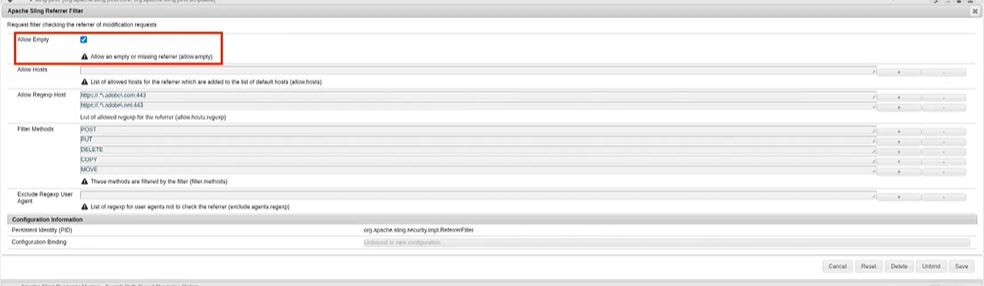
By default, AEM does not allow cross origin request from another domain. To allow the SPA application or third-party web-app to use AEM as headless service – we need to make following OSGi configurations

1. AEM CROSS ORIGIN RESOURCE SHARING POLICY (FACTORY CONFIGURATION)– This will allow the cross origin domain.



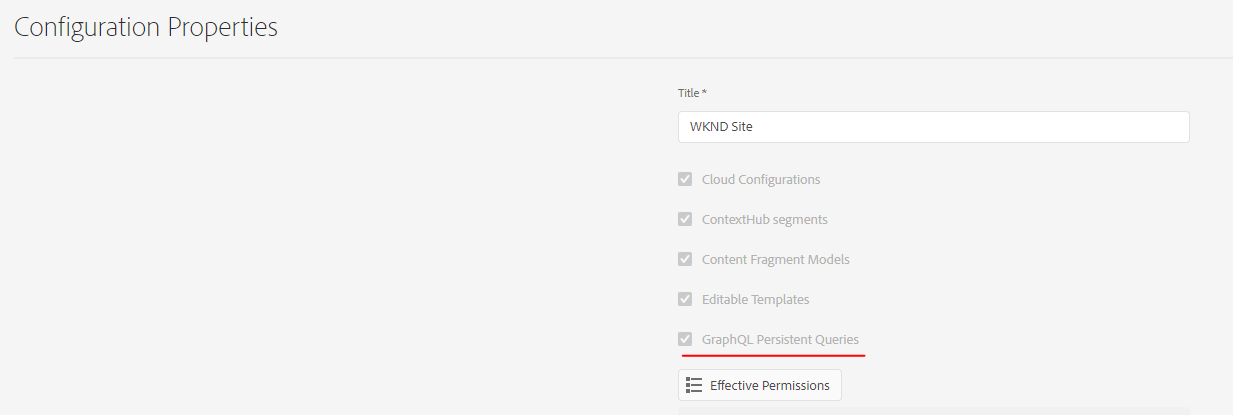
|  |  |  |
| --- | --- | --- |
| **CONFIGURATION** | **VALUE** | **DESCRIPTION** |
| **ALLOWED ORIGINS** | (Regex): http://localhost:.\* | Allows all local host connections. |
| **ALLOWED PATHS** | /content/graphql/global/endpoint.json | This is the only GraphQL endpoint currently configured. As a best practice, CORs configurations should be as restrictive as possible. |
| **ALLOWED METHODS** | GET, HEAD, POST | Only POST is required for GraphQL however the other methods can be useful when interacting with AEM in headless manner. |
| **SUPPORTED HEADERS** | Authorization | authorization has been added to pass in basic authentication on the author environment. |
| **SUPPORTS CREDENTIALS** | Yes | This is required as our React app will communicate with the protected GraphQL end points on the AEM Author service. |

1. APACHE SLIING REFERER FILTER: This will allow the empty to allow empty referrer headers



# GRAPH-QL PERSISTED QUERY

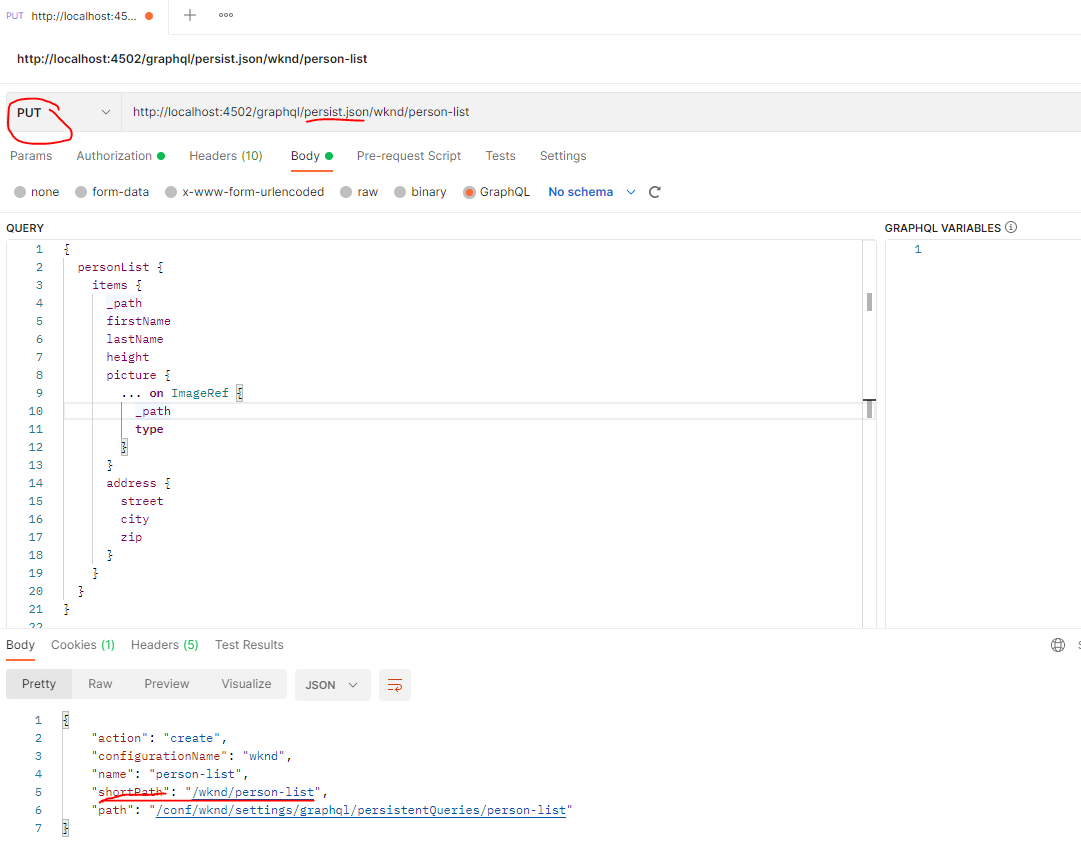
* To enable the graph query, navigate to Tool🡪 Configuration Browser
* Select the configuration folder of your project (we are using wknd site) 🡪 Properties
* Enable the “Graph QL Persistence Queries”



## CREATING THE PERSISTED QUERIES (PUT REQUEST)

* To create a persisted query in AEM – we have to send a post request to below end point.
* The request will create a node in AEM at /conf/<Site Name>/<Short Path>

|  |  |
| --- | --- |
| ENDPOINT- PUT REQUEST | EXAMPLE |
| http://localhost:4502/graphql/persist.json/<Site Name>/<Query Name> | <http://localhost:4502/graphql/persist.json/wknd/person-list> |

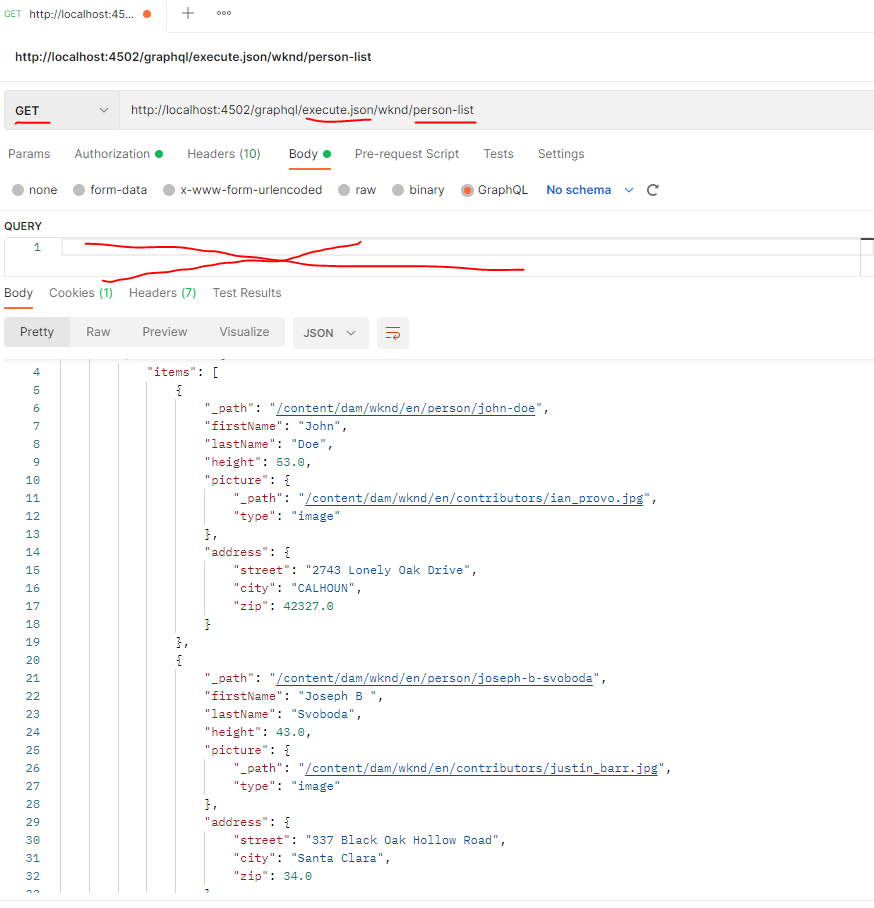


|  |  |
| --- | --- |
|  | * person-list is the *short path* used in the PUT request while creating the persisted query. * After the persisted query is saved in AEM(server side), We can do a get request to fetch the data – without sending anything in the request body. |

## EXECUTING THE PERSISTED QUERIES (GET REQUEST)

* As its Get request – we can be able to cache the response on dispatcher level.
* Note – We don’t have to send the anything in the request body.

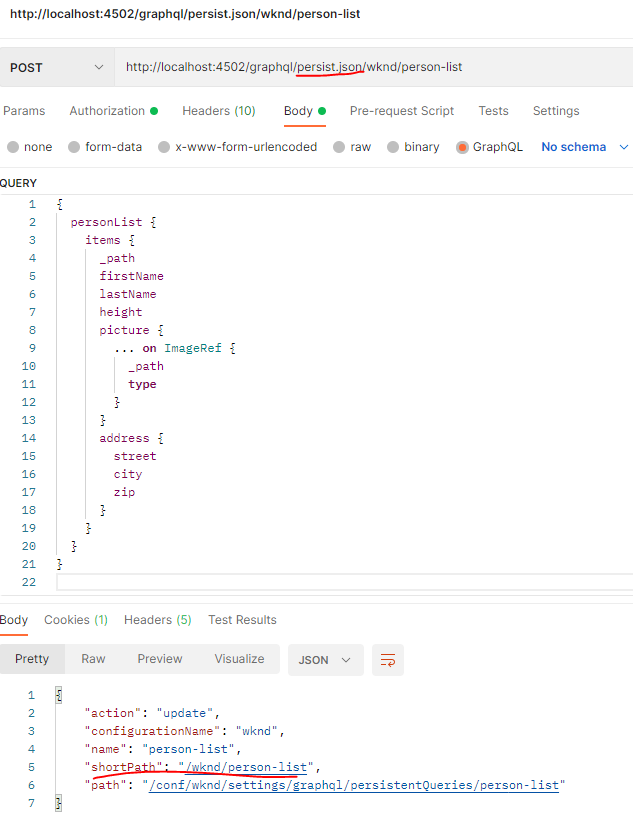
|  |  |
| --- | --- |
| ENDPOINT- GET REQUEST | EXAMPLE |
| http://localhost:4502/graphql/execute.json/<Site Name>/<Query Name> | <http://localhost:4502/graphql/execute.json/wknd/person-list> |



## UPDATING THE PERSISTED QUERIES

* If we want to update the persisted query, we can be able to update it using POST request.

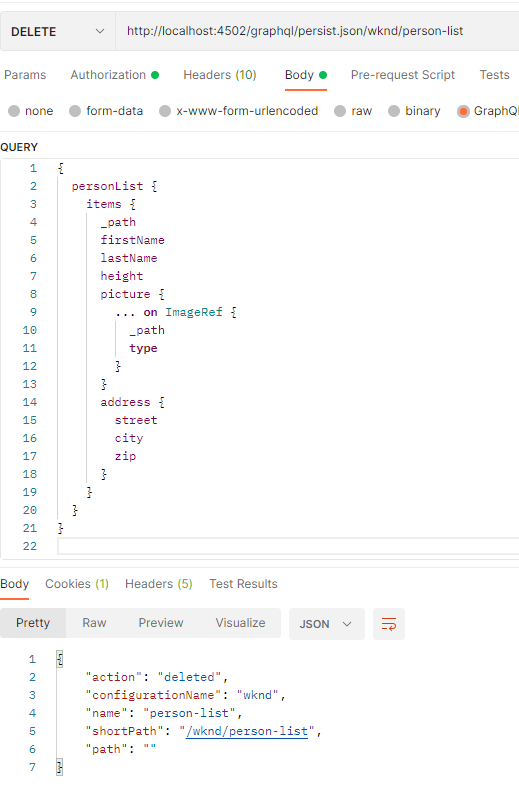
|  |  |
| --- | --- |
| ENDPOINT- POST REQUEST | EXAMPLE |
| http://localhost:4502/graphql/persist.json/<Site Name>/<Query Name> | <http://localhost:4502/graphql/persist.json/wknd/person-list> |



## DELETING THE PERSISTED QUERIES

* To delete the persisted query – we need to trigger a ***DELETE*** request to the same endpoint.

|  |  |
| --- | --- |
| ENDPOINT- DELETE REQUEST | EXAMPLE |
| http://localhost:4502/graphql/persist.json/<Site Name>/<Query Name> | <http://localhost:4502/graphql/persist.json/wknd/person-list> |



# EXECUTING THE PERSISTED QUERY USING CURL

