# The Beginning of Your GraphQL Journey with Magento 2.3

# Renato Cason

Tech Lead @ Bitbull

*@renatocason*

# Introduction

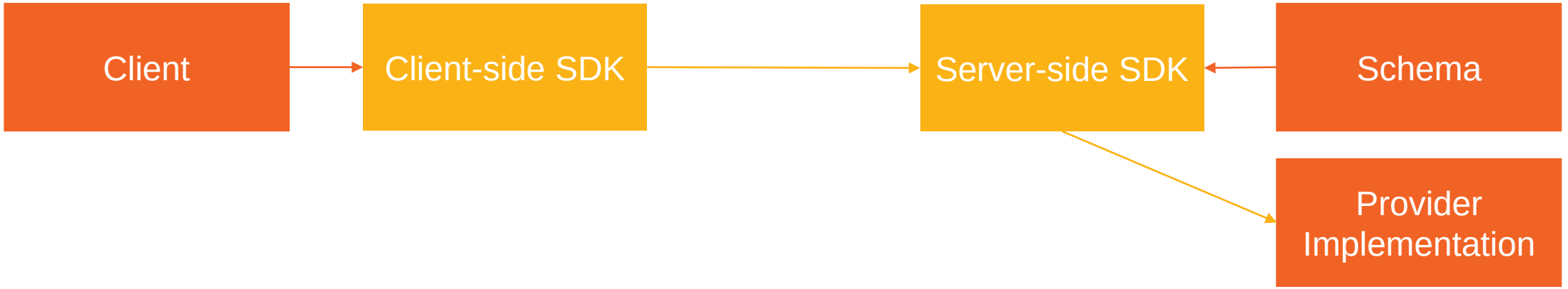
What is GraphQL?

# What is GraphQL?

https://graphql.org/

- Open-source data query and manipulation language for APIs
- Developed internally by Facebook in 2012
- Released as open source in 2015
- Moved to the GraphQL foundation in 2018
- Still under active development

# Structure




- Typically, both the client and server integration uses an SDK
- SDKs provide an abstraction layer from the communication protocol, although some actions might be supported or not
- Given the schema, the SDK takes care of serialization, deserialization, validation and of invoking the actual implementation of the provided functions

# What is good about GraphQL?

- Optimize network utilization
  - Specify the fields you need
  - Get multiple entities in one request
- Loose coupling between server and client
- Strong typing
- Developer tooling

# GraphQL vs REST

| GraphQL | REST | |
|---|---|---|
| Query | GET | Return an entity, or a list of entities |
| Mutation | POST / PUT / PATCH / DELETE | Creates, modifies or deletes an entity |
| Subscription | Webhooks | Listen to an entity’s real time updates |

# GraphQL query

```
{
  products(
    filter: { sku: { like: "24-WB%" } }
    pageSize: 20
    currentPage: 1
    sort: { name: DESC }
    ) {
      items {
      sku
    }
  }
}
```

# GraphQL mutation

```
mutation updateCustomerMutation($email: String!, $password: String!) {
  updateCustomer(
    input: {
      email: $email
      password: $password
    }
  )
  {
    customer {
      id
      email
    }
  }
}
```

# GraphQL in PHP

https://github.com/webonyx/graphql-php

composer require webonyx/graphql-php

```
$schema = new \GraphQL\Type\Schema([
    'query' => $queryType,
    'mutation' => $mutationType,
]);

// ...

$result = GraphQL::executeQuery($schema, $query);
```

# GraphQL in PHP - Query

```php
$queryType = new \GraphQL\Type\Schema\ObjectType([
    'name' => 'Query',
    'fields' => [
        'echo' => [
            'type' => Type::string(),
            'args' => [
                'message' => ['type' => Type::string()],
            ],
            'resolve' => function ($root, $args) {
                return 'Hello ' . $args['message'];
            }
        ],
    ],
]);
```

# Usage

```
# php -S localhost:8080 ./example.php &

# curl http://localhost:8080
   -d '{"query": "query { echo(message: \"World\") }" }'

{"data":{"echo":"Hello World"}}
```

# GraphQL in PHP - Mutation

```
$mutationType = new \GraphQL\Type\Schema\ObjectType([
    'name' => 'Calc',
    'fields' => [
        'sum' => [
            'type' => Type::int(),
            'args' => [
                'x' => ['type' => Type::int()],
                'y' => ['type' => Type::int()],
            ],
            'resolve' => function ($root, $args) {
                return $args['x'] + $args['y'];
            },
        ],
    ],
]);
```

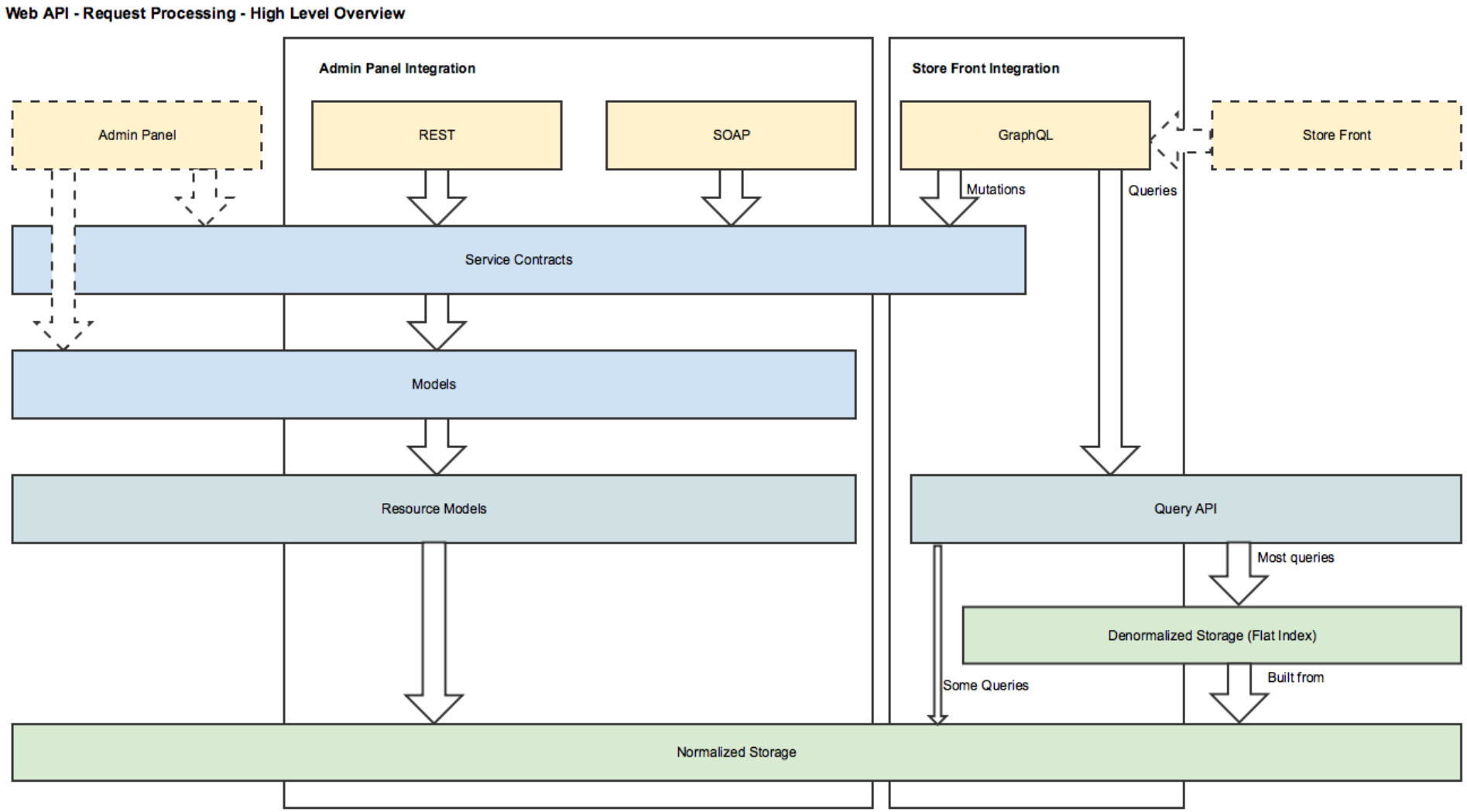
# Usage

```
# php -S localhost:8080 ./example.php &

# curl http://localhost:8080
   -d '{"query": "mutation { sum(x: 2, y: 2) }" }'

{"data":{"sum":4}}
```

# GraphQL in Magento

Current structure and future improvements.

# Architecture



# Magento 2.3.1 Code structure

Base implementation:

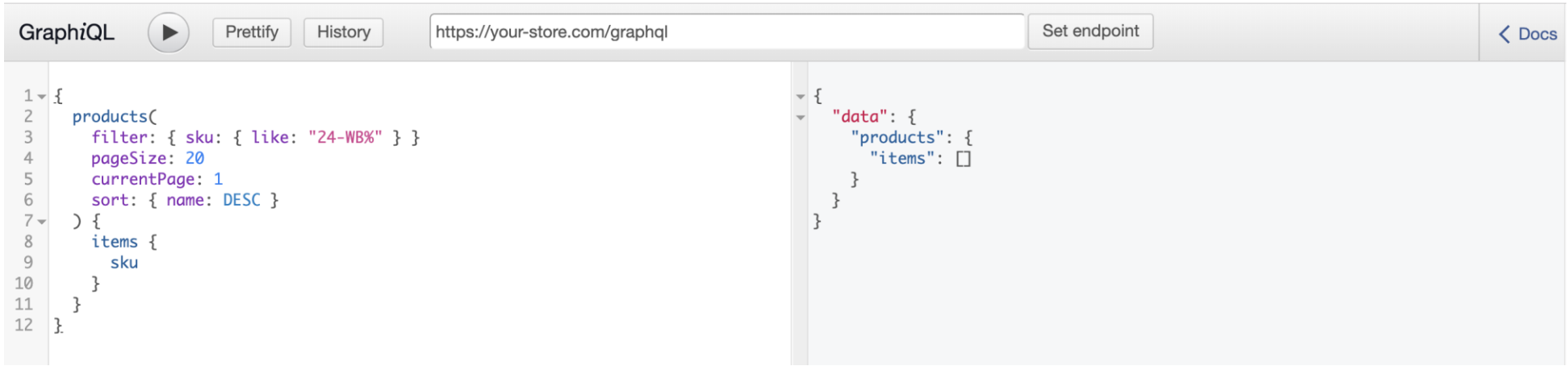
- Framework\GraphQl
- Framework\GraphQlSchemaStitching
- Magento\GraphQl

Extending core functionalities to provide queries and mutations:

- Magento_Customer
  - Magento_CustomerGraphQl
- Magento_Directory
  - Magento_DirectoryGraphQl
- …

# Usage

- Chrome ChromeiQL extension
- Access the */graphql* path on your store




*Note: accessing that url with a browser will cause an error, as only content type json is supported in the request*

# Magento 2.3.1 Support

- Store configuration
- Url rewrites
- All product types
  - Search and category
  - Layered navigation and filtering
  - Sort and pagination
  - Pricing
- Customer
  - Login/logout
  - Query and modify main account information and addresses
  - Wishlist
- Cart
  - Initialization and add simple/configurable products
  - Set shipping/billing address
  - Add/remove coupons

# Future developments

In Magento 2.3.2:

- Checkout
  - Offline shipping and payment method selection
  - Order placing
- MSI
  - Detailed stock information
- Performance improvements
  - Varnish caching

Following:

- Query API (GraphQL to SQL query)

# Customization

Implement GraphQL for your entities.

# Implement a custom Query or Mutation

- Add your custom GraphQL schema
- Implement a Resolver for your resource

Example
https://github.com/renatocason/imagine2019-storepicker-example/

**Warning**
The current GraphQL implementation is not *@api*:
minor upgrades could break your code.
GraphQL schema, on the opposite side, is *@api*.

# etc/schema.graphqls

```
type Query {
  locations: [Location]
    @resolver(class: "Rcason\\StorePicker\\Model\\Resolver\\Locations")
    @doc(description: "The locations query provides information for all supported
                       countries and the related stores.")
}

type Location {
  country_id: String
  store_id: String
  country: Country
}
```

# Model/Resolver/Location.php

- Extends *Magento\Framework\GraphQl\Query\ResolverInterface*
- Implements the *resolve* method
- It is not mandatory to use service contracts, collections and direct SQL queries are allowed

```
public function resolve(
    Field $field,
    $context,
    ResolveInfo $info,
    array $value = null,
    array $args = null
) {
    $searchCriteria = $this->searchCriteriaFactory->create();
    $locations = $this->locationRepository->getList($searchCriteria);
    $countries = $this->getCountriesById();

    $output = [];
    foreach ($locations as $location) {
        $output[] = $this->dataProcessor->buildOutputDataArray($location, LocationInterface::class);
    }

    return $output;
}
```

# Usage

```
{
  locations {
    country_id
    country {
      full_name_english
      full_name_locale
    }
    store_id
  }
}
```


```
{
  "data": {
    "locations": [
      {
        "country_id": "GB",
        "country": {
          "full_name_english": "United Kingdom",
          "full_name_locale": "Regno Unito"
        },
        "store_id": "1"
      },
      {
        "country_id": "US",
        "country": {
          "full_name_english": "United States",
          "full_name_locale": "Stati Uniti"
        },
        "store_id": "1"
      }
    ]
  }
}
```

# Testing

- Extend *Magento\TestFramework\TestCase\GraphQlAbstract*
- Rely on fixtures, sharing them across API interfaces

```
/**
 * @magentoApiDataFixture Magento/Customer/_files/customer.php
 */
public function testGetCustomer()
{
    $currentEmail = 'customer@example.com';
    $currentPassword = 'password';

    $query = <<<QUERY
query {
    customer {
        firstname
        lastname
        email
    }
}
QUERY;
    $response = $this->graphQlQuery($query, [], '', $this->getCustomerAuthHeaders($currentEmail, $currentPassword));

    $this->assertEquals('John', $response['customer']['firstname']);
    $this->assertEquals('Smith', $response['customer']['lastname']);
    $this->assertEquals($currentEmail, $response['customer']['email']);
}
```

# Caching and Performance

Keep your store fast.

# N+1 Problem

```
{
  customer {
    email
    firstname
    lastname
    addresses {
      street
      city
      country_id
    }
  }
}
```


```
{
  "data": {
    "customer": {
      "email": "renato.cason@gmail.com",
      "firstname": "Renato",
      "lastname": "Cason",
      "addresses": [
        {
          "street": [
            "Test 1"
          ],
          "city": "London",
          "country_id": "GB"
        },
        {
          "street": [
            "Test 2"
          ],
          "city": "London",
          "country_id": "GB"
        }
      ]
    }
  }
}
```


- In the example, we are loading one customer, and two addresses
- If they are fetched individually from the database, this will result in 3 queries (N+1: 2+1)

*Usually resolved with batching, which in this case would result in 2 queries.*

# Caching

https://graphql.org/learn/caching/

- In endpoint-based APIs, HTTP caching can be used
  *i.e. Varnish or similar, plus CDN*
- Providing Object Identifiers allows clients to build rich caches
  *needs the client to be smart, and implement the caching logic*

# HTTP Caching

*Can* work, but might not be *as* efficient

Client A:

```
{
  products(
    filter: { sku: { like: "24-WB%" } }
    pageSize: 20
    currentPage: 1
    sort: { name: DESC }
  ) {
    items {
      sku
      name
    }
  }
}
```

Client B:

```
{
  products(
    filter: { sku: { like: "24-WB%" } }
    pageSize: 20
    currentPage: 1
    sort: { name: DESC }
  ) {
    items {
      name
      sku
    }
  }
}
```

# HTTP Caching

*Can* work, but might not be *as* efficient

Client A:

```
{
  products(
    filter: { sku: { like: "24-WB%" } }
    pageSize: 20
    currentPage: 1
    sort: { name: DESC }
  ) {
    items {
      sku
      name
    }
  }
}
```

Client B:

```
{
  products(
    filter: { sku: { like: "24-WB%" } }
    pageSize: 20
    currentPage: 1
    sort: { name: DESC }
  ) {
    items {
      name
      sku
    }
  }
}
```

# Caching – Server side

Use Magento’s built in caching tools to bypass database access.

Pros:

- Relatively easy to implement

Cons:

- Not applicable to all resource types
- Can make Redis become a bottleneck

# Caching – Client side

Use the client's library to cache the data.
i.e. https://www.apollographql.com/docs/react/advanced/caching

Pros:

- Totally bypasses the network

Cons:

- Needs to be implemented for each client (i.e. web and mobile)
- Doesn't scale for multiple users

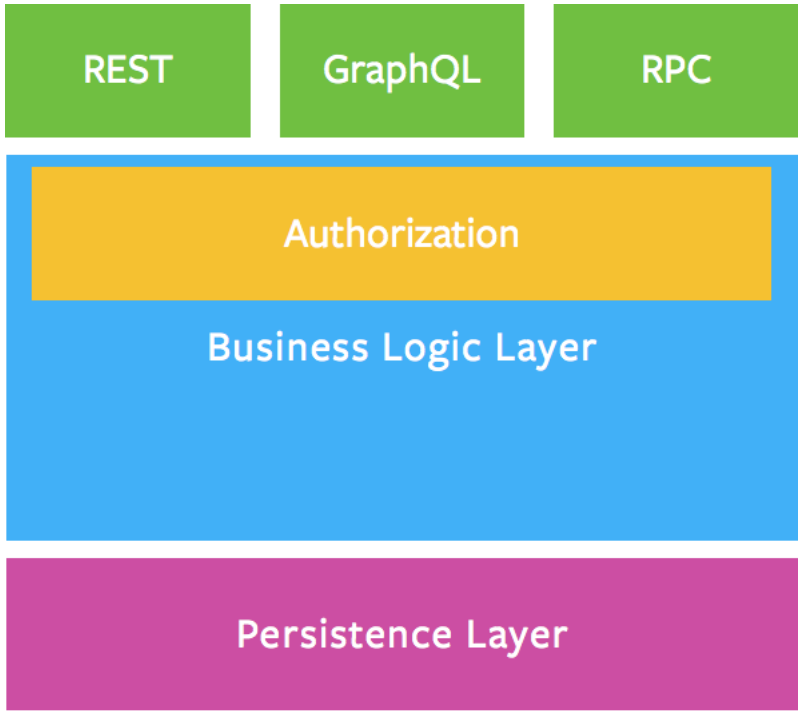
# Authorization

Secure your queries and mutations.

# Authentication and Authorization

https://graphql.org/learn/authorization/

- Delegate authorization logic to the business logic layer
- Your business logic layer should act as the single source of truth for enforcing business domain rules



# Authentication

https://devdocs.magento.com/guides/v2.3/graphql/get-customer-authorization-token.html

- Only customer authentication is supported
- No plan in the short/mid term to support admin authentication

# Authentication

```
mutation {
  generateCustomerToken(email: "customer@example.com",
                                    password: "password") {
    token
  }
}
```

```
{
  "data": {
    "generateCustomerToken": {
      "token": "hoyz7k697ubv5hcpq92yrtx39i7x10um"
    }
  }
}
```

# Authorization

Uses a Bearer Token, exactly as REST does

Example:

*Magento\QuoteGraphQl\Model\Resolver\CreateEmptyCart*

```
public function resolve(…) {
    $customerId = $context->getUserId();
```

# Conclusion

Wrapping up.

# Conclusion

- Simple to learn
  - Many online resources
  - Easy syntax
  - Great developer tools
- Main advantages are for building highly flexible APIs
  - Suitable for customer facing functionalities
  - Not ideal for ERP integrations
- Caching can be complex
  - If you need to scale, design your application to that from the beginning

# Thank you.

any questions?

# imagine

2019