User Guide

Public API

May 2024

1-800-488-1803
https://www.ezcater.com/

Contents

Introduction to the Public API

An API (**A**pplication **P**rogramming **I**nterface) is a type of software that facilitates the secure transfer of data between other pieces of software.

ezCater has built systems to support integration between your brand's ezCater orders and other platforms, such as Point of Sale platforms or reporting systems, including a webhook event system and a publicly consumable API to retrieve Menu and Order data.

This guide is intended to provide an overview of the processes involved in setting up the Public API's functionality. It also provides examples of some of the ways you can use these tools to pull information from ezManage and into your POS.

Overview of Setup

This document will go into more detail below as to how your connection to the Public API will be implemented. In a general sense, you'll need to:

1. Create an ezManage user specifically for your integration
2. Generate a one-time authentication token for that user
3. Use that token to connect to the API
4. Create a subscriber with GraphQL
5. Use that subscriber to set up any desired subscriptions

What kinds of tasks can the Public API be used for?

The Public API utilizes webhooks to allow you to pull information on order events and on your stores' menus into your Point of Sale or other integrated platforms. The exchange of information is one-directional; the API is not built to accommodate any injection of information back into ezManage. By integrating with the API, you're able to populate order information for injection into a store's POS, with no manual entry required.

How does this integration affect order acceptance?

The short answer is, it doesn't!

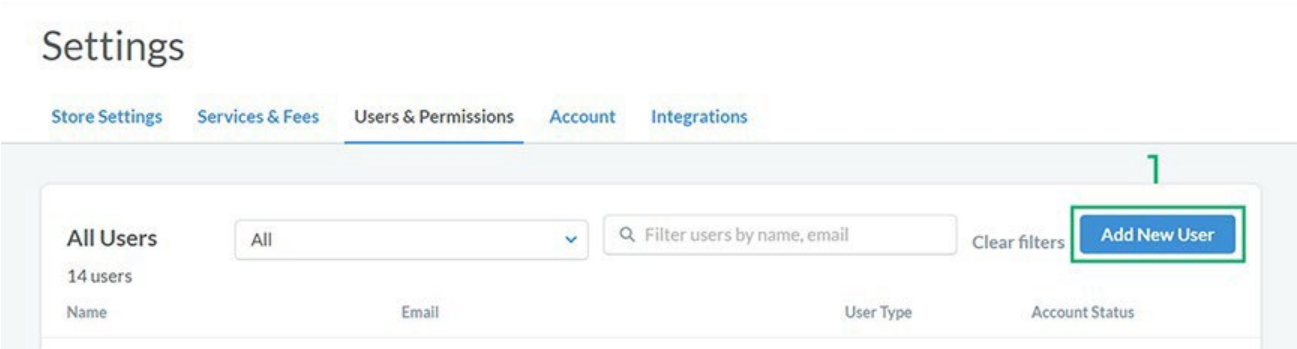
Since the Public API is not designed to pass information or updates back into ezManage, it isn't used to accept orders. Your workflow for order acceptance will continue the same way that you're used to. It's also important to keep in mind that the subscriptions we'll detail further down in the document send a webhook when an order is accepted - not when it's placed. Integration with the API has no effect on your current setup of order notifications; you'll still get the phone calls, emails, and/or text messages alerting you to new ezCater orders just like you're used to.

Using the Public API

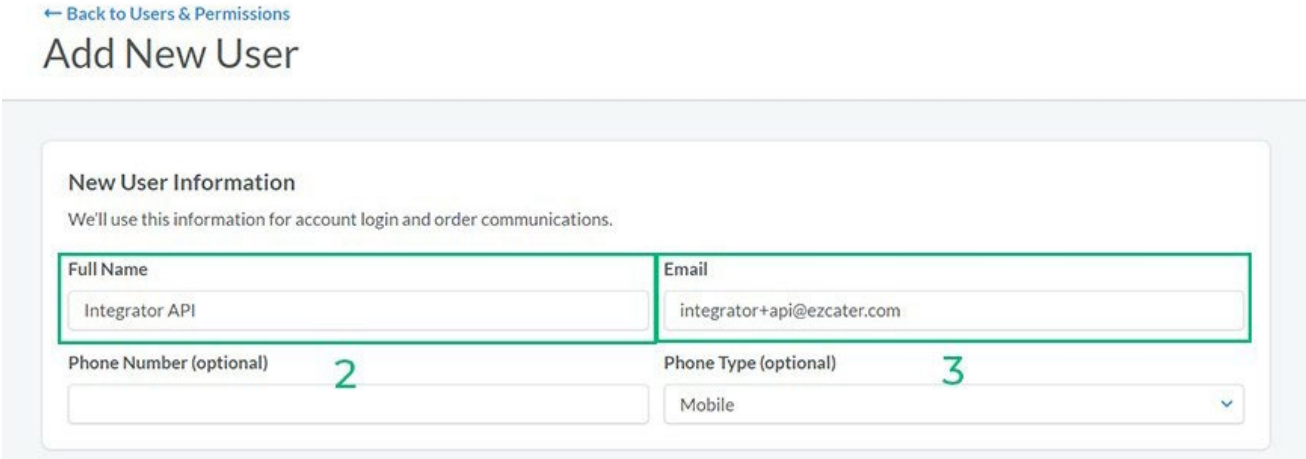
How to Create API Users in ezManage

The first step in connecting to the Public API is to create a specific user for it within ezManage. You will need to be logged in as an Admin user to grant the needed permissions.

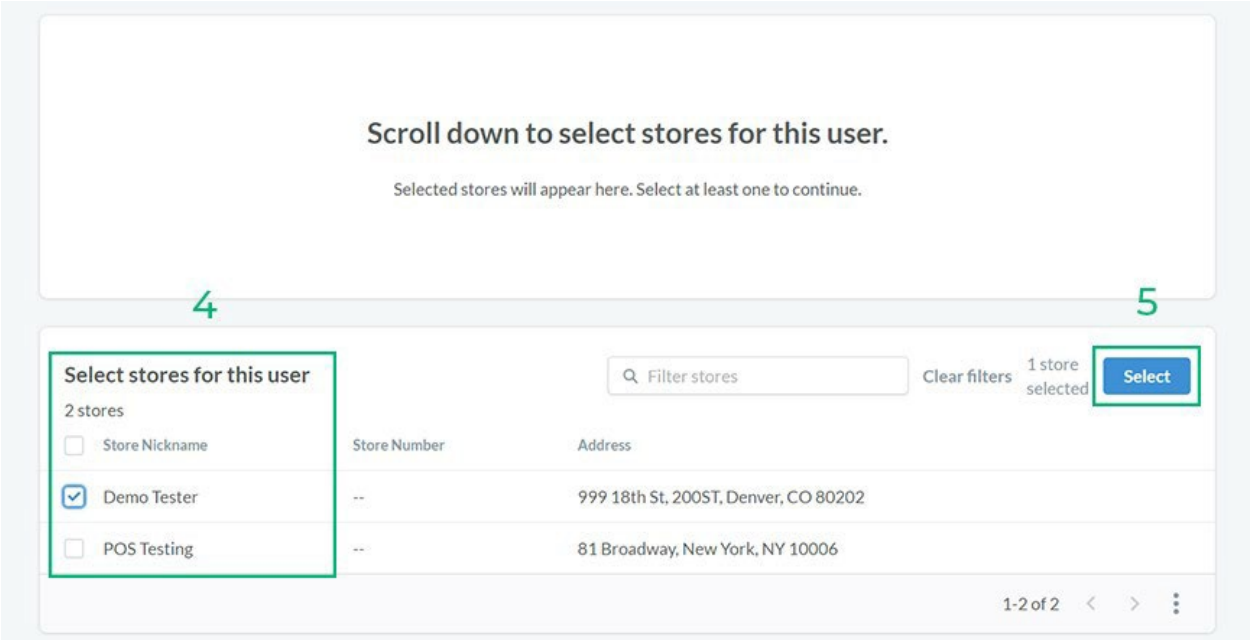
1. Go to "Users & Permissions" within the Settings tab of ezManage, then select "Add New User."



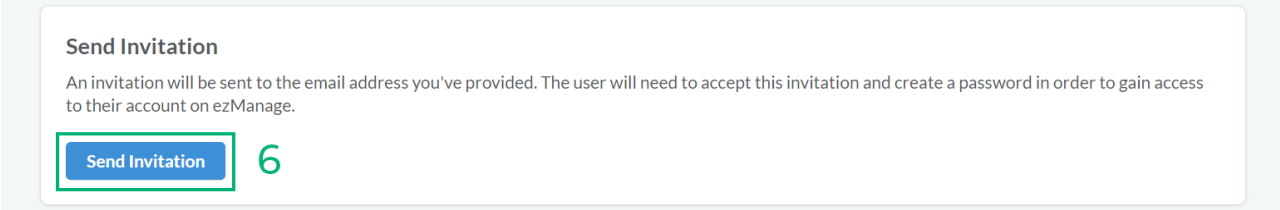
2. Input key information about the user. If you are using a third party integrator, we recommend that the first name be the integrator (for example: Checkmate, QuBeyond) that you are working with and the last name being API.
3. For the email, please ask your integrator to create a unique email address for your brand, which they can receive the ezManage invitation at and log in to generate the API Token. Most integrators use a format similar to: **ezcater+brand@integratordomain.com** depending on their email provider. If you are not using a third party integrator, or are unable to use the recommended email format, please reach out to api_support@ezcater.com to discuss potential alternatives.



4. Next, select which stores your integration should have access to. All locations associated with the user will be available to the integration. **Note:** These users should have access to at least one location.
5. Once the locations have been selected, click "Select". The stores you selected will populate the table.

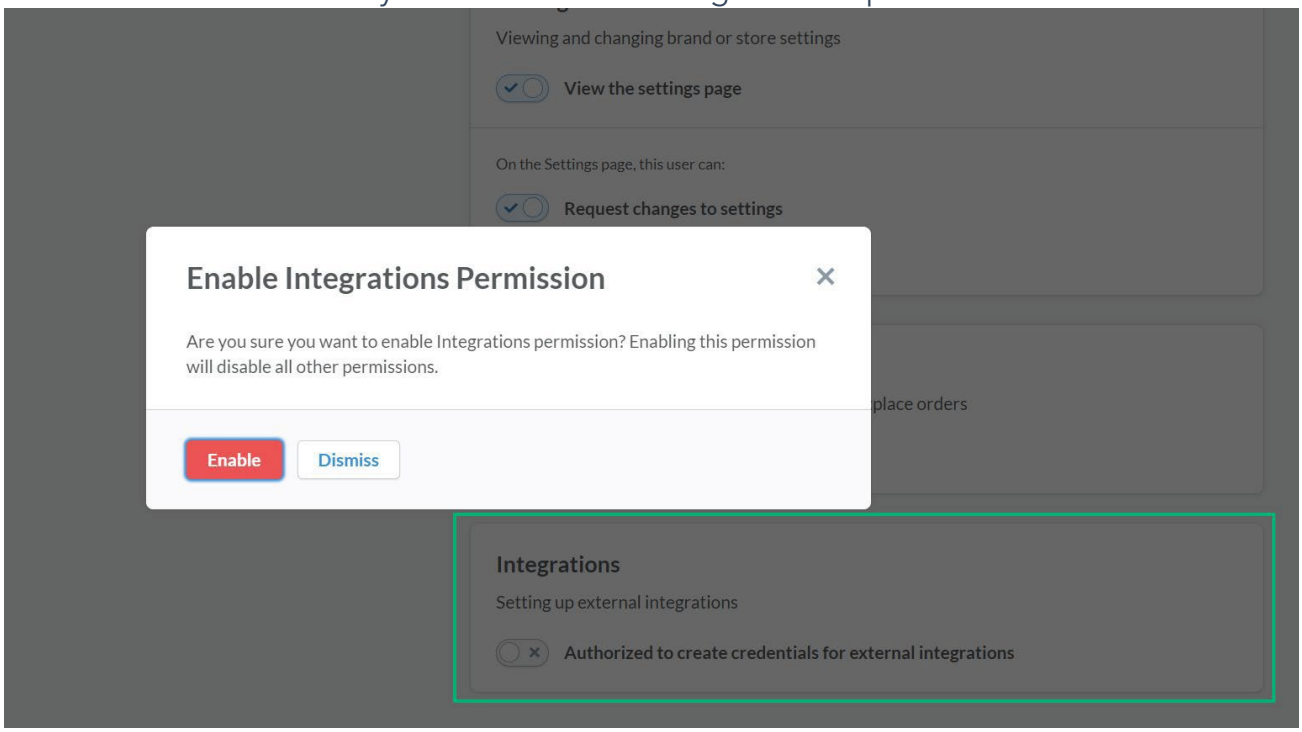


6. Click "Send Invitation" at the bottom of the page to create your API user.



Once the integration partner has accepted the invitation, you will be able to grant them the permission needed to generate their integration credentials.

1. Go to "Users & Permissions" within the Settings tab of ezManage, then select the API contact you created.
2. Once on the API contacts page, select the “Permissions” tab from the top navigation.
3. Scroll to the bottom and you should see an “Integrations” option.



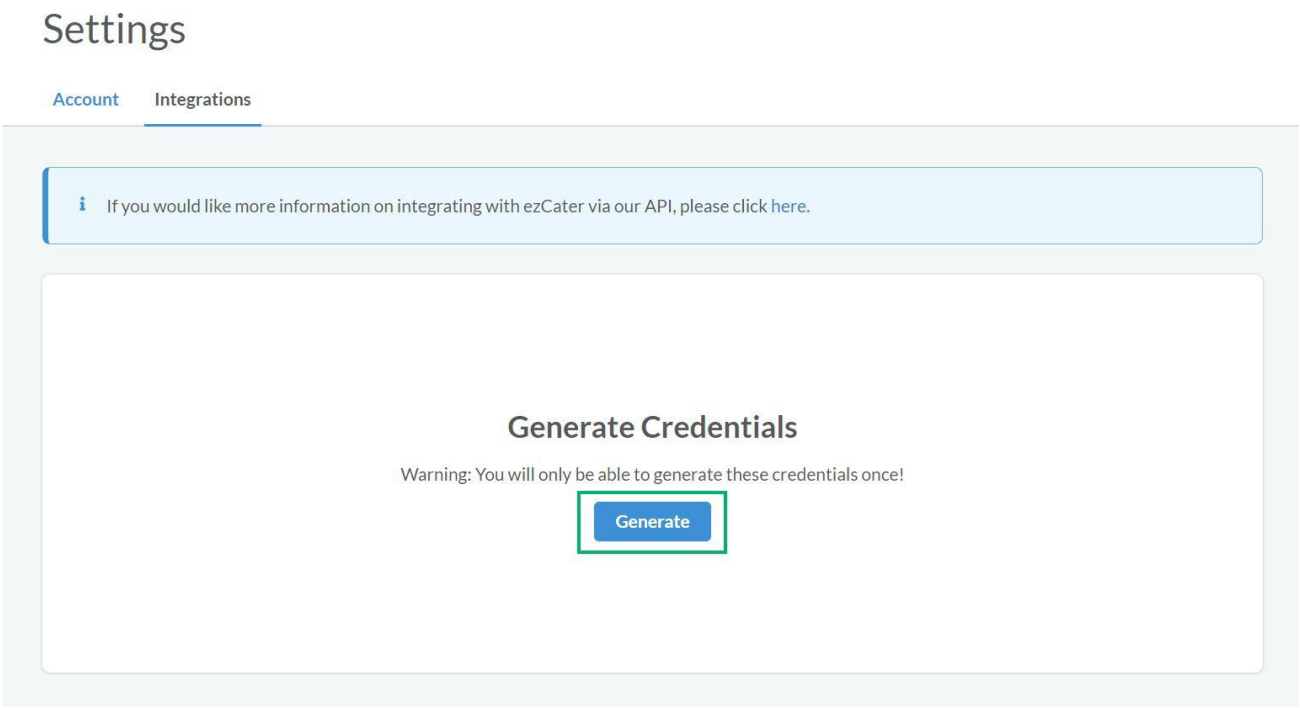
4. Toggle on “Authorized to create credentials for external integrations”.

Note: Granting Integrations permission will disable any other permissions for this user.

How to Get Your API Token

Once your user profile has been created and permissions have been granted by the catering partner, your technical specialist can acquire the authorization token.

1. Sign into the API User account that was created.
2. Go to Integrations within the Settings tab of ezManage.



3. Click "Generate" to receive your unique authorization token.

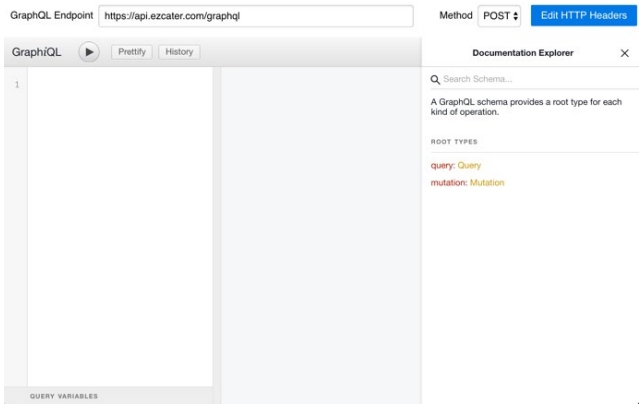
Make sure to save this token! The token can only be granted once, and if lost cannot be recovered. This token will be used as you set up the integration on our Public API.

Using GraphQL

The remainder of this guide assumes that your team is familiar with the use of API Clients, and are able to set up a connection via a client of your choice. All requests must be made to our public GraphQL endpoint at https://api.ezcater.com/graphql via HTTP POST with an 'Authorization' header using the generated token as its value.

If you have never used GraphQL before, there is a lot of good information and examples to be found at https://graphql.org/learn

GraphQL leverages a pattern called introspection, which allows you to query the endpoint for information about the schema and structures you can request. Tools like GraphQL will handle this automatically, providing an easy to navigate documentation that you can use to see what queries are available, what fields you can ask for and what the return values will look like.



Building a Request

Make a POST request with your Query body to https://api.ezcater.com/graphql, using the following **headers**:

- **Content-Type**: application/json
- **Authorization**: <Your API Token>
- **Apollographql-client-name**: <Your Organization Name/Identifier>
- **Apollographql-client-version**: <Your Software Version>

In order for ezCater to properly track and troubleshoot requests to our API, all requests *must* be named. A collection of our naming conventions for each query can be found at the bottom of this document.

Template:

```
query <NAMEOFQUERY> {
    Menus {
        Nodes {
            Id
            Name
            startDate
            endDate
        }
    }
}
```

Example:

```
query menusByCaterer {
    menus {
        nodes {
            id
            name
            startDate
            endDate
        }
    }
}
```

As an additional example, you can query __schema to list all types defined in the schema and get details about each:

```
query fullSchema {
    _schema {
        types {
            name
            kind
            description
            fields {
                name
            }
        }
    }
}
```

Or query __type to get details about any type:

```
query type {
    __type(name: "Menu" ) {
        name
        kind
        description fields {
            name
        }
    }
}
```

Setting Up Your Integration

Once you have your API Token and have connected to our endpoint via an API Client, you can begin setting up your integration with the Public API.

Creating a Subscriber

A **Subscriber** represents the integration itself. The **Subscriber** will manage where event notifications are sent to as well as what events you want to subscribe to. **NOTE:** The **webhookURL** is the location you want the event notifications to be sent to.

Create Subscriber Mutation

To create the **Subscriber**, simply make a **createSubscriber** mutation request to the GraphQL endpoint.

Example Mutation:

```
mutation createSubscriber{
    createSubscriber(subscriberParams:{ name:
        "Sample Integration",
          webhookUrl:  "http://your-webhook-url.com/your-desired-endpoint"
        }){
            subscriber{ id
            name webhookUrl
            webhookSecret
        }
    }
}
```

Make sure you include the **webhookSecret** (line 10) in the values you want to return, and save it! The **webhookSecret** can only be seen on Subscriber creation, you will not be able to query for it afterwards.

The response will look something like this:

```
{
    "data":{
        "createSubscriber":{ "subscriber":{
            "id":"0fc73833-fac0-4a4b-be38-b0a9baea4507", "name": "Sample
            Integration",
            "webhookUrl":
              "http://your-webhook-url.com/your-desired-endpoint", "webhookSecret":
              "A9e8018dac74463c632dea294494ca468f470c4df8513309d0e7ef8e265e 08f7"
        }
    }
    }
}
```

NOTE: We currently only support creating one subscriber per API user. If you are seeing errors that your subscriber could not be created, it may be that one has already been set up. Try running the subscribers query to see if any already exist. **TIP:** You will also need the Subscriber ID in subsequent steps, so we recommend keeping track of it now.

Creating Your Subscriptions

Subscriptions are how the **Subscriber** determines what events it needs to send out notifications for. Once you've created your subscriber, you can customize the sorts of information that you may want to pull into your POS systems.

NOTE: We currently are only supporting **Order Accepted, Order Cancelled** and **Menu Updated** subscriptions.

Caterer Query

To create your subscriptions you will need to know the **UUID** of the locations for which you want notifications to be sent across the integration. These UUIDs can be retrieved by making a **caterers** query to the endpoint.

EXAMPLE QUERY:

```
query allCaterers {
    caterers {
        uuid
        name
        storeNumber
        address {
            street
            street2
            city
            state
            zip
        }
    }
}
```

It may be useful to include the address information so that you can know which UUID corresponds to which location.

Create Subscription Mutation

Now that you have your location **UUID** you can begin to create your subscriptions using the **createSubscription** mutation. For each location, you will need to send two mutations (one for accepted events and one for cancelled events). As the integration functionality expands, there may be additional subscription options as well.

As previously mentioned, the order accepted subscription triggers once an order has been accepted, not when the order is placed. It also triggers when an order has been updated & that update is accepted.

We recommend exploring the schema so that you can determine which available values are going to be useful for meeting your needs. We've provided some examples of how these subscriptions might look below.

NOTE: If you need your **subscriberId** you can run the **subscriber query**:

```
query allSubscribers{
    subscribers{ id
    }
}
```

EXAMPLE MUTATIONS:

Create an order accepted subscription:

```
Mutation createSubscription {
    createSubscription(subscriptionParams: { subscriberId:
        "your-subscriber-id", eventKey: accepted,
        eventEntity: Order, parentEntity: Caterer,
        parentId: "your-caterer-id"
    }) {
        subscription{
        parentEntity
        parentId eventKey
        eventEntity
        }

    }
}
```

Create an order cancelled subscription:

```
mutation createSubscription{
    createSubscription(subscriptionParams: { subscriberId:
        "your-subscriber-id", eventKey: cancelled,
        eventEntity: Order, parentEntity: Caterer,
        parentId: "your-caterer-id"
    }) {
    subscription {
        parentEntity
        parentId eventKey
        eventEntity
        }
    }
}
```

Create a menu updated subscription:

```
mutation createSubscription{
    createSubscription(subscriptionParams: { subscriberId:
        "your-subscriber-id", eventKey: updated,
        eventEntity: Menu, parentEntity: Caterer,
        parentId: "your-caterer-id"
    }){
    subscription{
        parentEntity
        parentId eventKey
        eventEntity
        }
    }
}
```

Once your subscriptions are set up, you should now receive Event Notifications at the **webhookUrl** that was specified during **Subscriber** creation.

NOTE: If you need to change the **webhookUrl**, you can use the **updateSubscriber** mutation to change the **webhookUrl** and/or **name** of your **Subscriber**.

Adding Your External IDs

ezCater's API Support team will add your menu's External IDs/PLUs to the ezCater menu for use in querying. The team will export your menu; you will fill in the PLUs for your menu items; and the team will import the information back into the ezCater menu. To initiate this process, contact api_support@ezcater.com

Optional Setup: Validating the Webhook

You have the option to validate whether the webhook you received actually came from ezCater.

The X-Ezcater-Signature header value consists of a timestamp and the signature separated by a period. You can use the timestamp from this header, your webhookSecret value (from when you initially created the subscription), and the request body to verify this signature.

1. **Create a computed signature payload from the webhook request data**

   You need two pieces of information for this step:

   - The timestamp from the X-Ezcater-Signature header, which is the first portion before the period
   - The request body

   Concatenate these two values using a period to obtain a computed signature payload. For example, in Ruby this would look something like:

   ```
   timestamp = x_ezcater_signature_header.split(".")[0] computed_signature_payload = [timestamp.to_i, request.body].join(".")
   ```

2. **Compute an HMAC signature of the request data**

   You need two pieces of information for this step:

   - The computed signature payload from Step 1
   - The webhook secret provided to you when you created your subscription

   Compute an HMAC signature using your webhook secret and the computed signature payload. For example, in Ruby this would like something like:

   ```
   Signature = OpenSSL::HMAC.hexdigest("sha256", webhook_secret, computed_signature_payload)
   ```

3. **Compare the provided signature with the computed one**

   You need two pieces of information for this step:

   - The computed HMAC signature from Step 2
   - The provided signature from the webhook request

   Compare the computed HMAC signature with the value after the period in the X-Ezcater-Signature header. If these two values match, the request is valid. If they do not match, the request may have been tampered with or originated from an unauthorized source.

Querying the Public API

We recommend using a tool like GraphiQL for viewing all the fields that can be queried from the Public API. This section will give a brief overview of our top-level queries.

Caterers Query

The **Caterers** query returns basic information about all of the catering locations that have been tied to the integration. This query is typically used to find **Caterer UUIDs** for creating **Subscriptions** and querying for a specific **Menu**.

EXAMPLE QUERY:

```
query allCaterers {
    caterers {
        uuid
        name
        storeNumber
        address {
            street
            street2
            city
            state
            zip
        }
    }
}
```

It may be useful to include the address information so that you can know which UUID corresponds to which location.

Order Query

When you receive an Event Notification, the payload will include the **UUID** of the **Order** that has changed.

You can use this **UUID** to query the Public API to see more details about the **Order**. Note that you must have permission within ezManage to access the store associated with an order UUID to be able to pull information about it.

EXAMPLE QUERY:

```
query orderById {
    order(id: "UUID") {
        orderNumber
        orderSourceType
        isTaxExempt event {
            headcount
            timestamp
            catererHandoffFoodTime orderType
            thirdPartyDeliveryPartner
     }
    caterer {
        uuid
            live  name
            address{
                street
                city state
            }
    }
    totals {
        customerTotalDue{
            currency
            subunits
        }
        salesTax {
            currency
            subunits
        }
        salesTaxRemittance{
            currency subunits
        }
        subTotal {
            currency
            subunits
        }
        tip {
            currency
            subunits
        }
     }

    catererCart{
        totals {
            catererTotalDue
        }
        feesAndDiscounts(type: DELIVERY_FEE){
```

```
                    _typename
                    name
                    cost{
                        currency
                        subunits
                    }
                }
                orderItems{
                    name
                    uuid
                    totalInSubunits{
                        subunits
                        currency
                    }
                    menuId posItemId
                    menuItemSizeId
                    quantity
                    noteToCaterer
                    specialInstructions
                    customizations{
                        customizationId
                        posCustomizationId
                        name
                        quantity
                        customizationTypeName
                        }
                    }
                }
            }
        }
```

Subscriber Query

The **Subscriber** query returns information about the **Subscribers** and **Subscriptions** for the integration. This query can be useful in determining if you have successfully created a **Subscriber** or **Subscription.** At this time we only allow one Subscriber per API user.

EXAMPLE QUERY:

```
Query allSubscribers{
    subscribers{ id
        name
        webhookUrl
        subscriptions{
            parentId
            parentEntity
            eventEntity
            eventKey
        }
    }
}
```

Event Notifications

Once you have set up your integration, you will begin to receive **Event Notifications** for any **Subscriptions** you have set up. Notifications will be sent to the **webhookUrl** specified during **Subscriber** creation. The notifications will provide basic information about the event that occurred, but for detailed information, you can query the Public API.

EXAMPLE NOTIFICATION PAYLOAD:

```
{
    "id": "[uuid of the notification]", "parent_type":
    "Caterer", "parent_id": "[uuid of the caterer]",
    "entity_type": "Order",
    "entity_id": "[uuid of the order]", "key":
    ["accepted",]OR["cancelled",] "occurred_at":
    "[time]"
}
```

NOTE: At this time we are only supporting **Order Accepted** and **Order Cancelled** notifications. We will communicate changes as more notification types are added.

Troubleshooting the API

If your IT Department runs into issues that need troubleshooting when using the Public API, remember that you can always reach out to ezCater's API support team at api_support@ezcater.com

In general, any errors in a request will be passed back in the response with a non- empty "errors" key.

```
{
        "data": { ... },
        "errors": [ ... ]
}
```

The contents of the errors key will provide information on what is wrong. The format will follow the GraphQL specification.

Data Dictionary

Mutations

Function: Subscriptions
Description: Available subscription mutations

Subscription	Description
Order Accepted	Notifies subscribers when an order has been accepted
Order Cancelled	Notifies subscribers when an order has been cancelled

Queries

Function: Queries
Description: Callable queries

Query	Description
caterers	Information about all locations tied to this integration
order	Details for this specific order
subscriber	Information about all subscribers and subscriptions for this integration

Query Naming Conventions

Function: Naming Conventions

Description: Naming standards for queries and mutations

Request	Name
caterers	allCaterers
order	orderByID
subscribers	allSubscribers
menu	menuByID
menus	menusByCaterer
__schema	fullSchema
__type	type
createSubscriber	createSubscriber
updateSubscriber	updateSubscriber
createSubscription	createSubscription