SambaPOS Club

# Integrator's GraphQL API Guide

V5 Tutorial

emre 2017-02-15 10:16:11 UTC #1

For this tutorial you need to install at least **SambaPOS 5.1.62** version.

If you're interested to create tickets or fetch menus through your application this tutorial will help you to setup SambaPOS and prepare your GraphQL scripts.

GraphQL is the name of the HTTP based API you'll use to access SambaPOS and execute queries. By default SambaPOS does not allow executing GraphQL queries externally. You need to setup SambaPOS Message Server to allow that.

Message Server is the name of server application that runs on server machine to notify SambaPOS terminals when new tickets are created. It works on two modes.

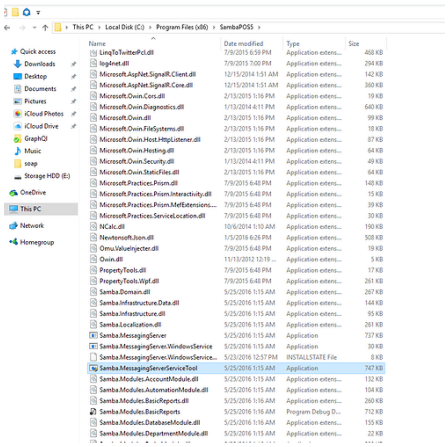
1. **Notification Only.** This is default mode. It only notifies terminals to refresh terminal screens when another terminal creates or updates a ticket.
2. **API access.** Besides notifying terminals it allows executing GraphQL queries. This mode also allows terminals other than SambaPOS terminals to subscribe to "new ticket creation" notifications.

To enable API Access you need to:

1. Setup Server Application to enable API access.
2. Setup SambaPOS application itself to subscribe to notifications.
3. Setup Authorization to allow third party service to access GraphQL API.

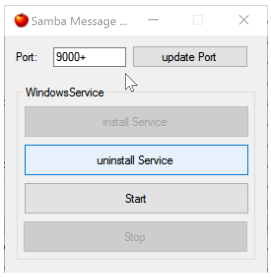
## Server Setup

On server start `Samba.MessagingServerServiceTool.exe` application to bring up service setup helper tool.



Pasted image851x846 80.8 KB

After setting up desired port add "+" sign at the end of port number and click `Update Port` button. Having + char at the end of the Port Enables API mode.



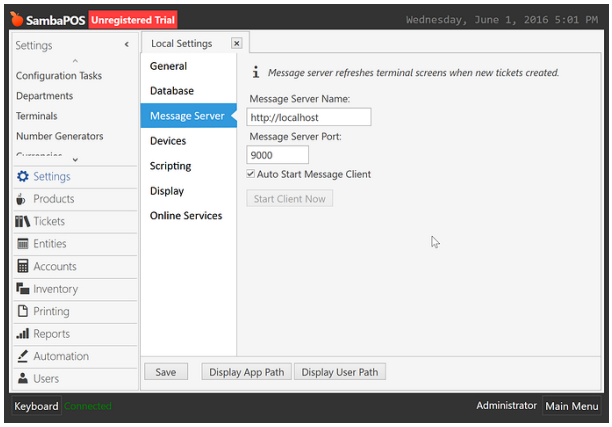

When you click on Update Port button it should start the service automatically. If it does not click "Start" button to start service.

## SambaPOS Setup

To subscribe SambaPOS to notifications you'll make a simple change on local settings page.

1. Start SambaPOS on a terminal.
2. `On Management > Local Settings > Message Server` screen, type the name of the server, port and enable Auto Start Message Client option.
3. Restart SambaPOS.

You need to do that for all terminals.



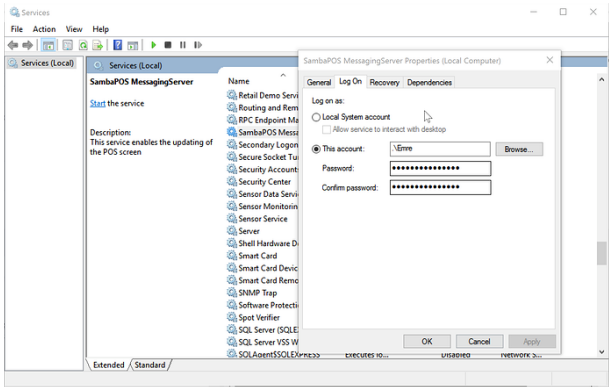

Pasted image1863x1289 117 KB

Follow these to ensure your setup is correct.

1. The screen shot shows server name as **localhost**. Instead of `localhost` you need to type server name or server IP address. For example if your server name is `MyServer` you need to type `http://MyServer` as server name.
2. While typing server name you need to add “http://” in front of it.
3. Don’t add + sign at the end of port.
4. Be sure your firewalls allows communication through these ports. When firewall prompts allow communication or configure Windows Firewall to open these ports. If you already running message server on `Notification Mode` no additional setup is needed.

## Remote Access

For now server allows running queries only on server. If you need to allow access from other devices you need to run Message Server service as a user that have Administrator privileges.




Pasted image1866x1184 80.2 KB

Please ensure firewall properly configured to allow communication through ports and also service application allowed through firewall. If it works through localhost address but you receive no response when you attempt to connect remotely this is probably a firewall configuration issue.

## Server Database Access

Message server uses connection string taken from the LocalSettings.txt file. So you need to run SambaPOS on Server PC, setup database and restart server to update database connection settings. Please keep in mind if you’re using SQL Server LocalDB as the database engine you need to start Message Server Service as the user that actually logged on the PC. LocalDB stores database files under User’s My Documents folder and if you start service with a different user it will access to a different database. If you’re using SQL Express and can connect to database from this PC no additional configuration is needed.

On next part we’ll setup Authorization.

---

Authorization has been denied for this request

---

Is Mobile Client for use over the internet or in house wireless network?

---

SambaPOS 5.1.62 Release

---

Welcome to SambaPOS 5

---

**emre** 2017-02-15 10:16:16 UTC #2

---

**emre** 2017-02-15 11:24:57 UTC #3

## Authentication

Before going into details about authentication I’ll try to explain how it works in general. We assume the server name is **server** and it runs on **9000** port. While applying tutorial you need to change them for your setup.

When Message Server runs on `API Access` mode it starts serving an endpoint through `http://server:9000/api/graphql` URL. When you post a query to the URL server responds with the result of the query.

To be able to execute a graphql query you need to append a security key to your query request. This key is called **access token** and gets sent to server via request header with Authorization key.

image.png1682x1137 237 KB

Without a valid access token GraphQL Server will respond with a **401 Unauthorized** Error Message.

To be able to obtain an access token you need to request one from http://server:9000/Token URL with a `user name` and a `password`.

To configure a user name and password we'll use existing SambaPOS users. We have a new `password` field for Users. By default no users have a password so we need to set a password for a user to be able to test it. Unlike pin codes passwords expected to be stronger but for this demo I'll set 1111 password for Administrator.

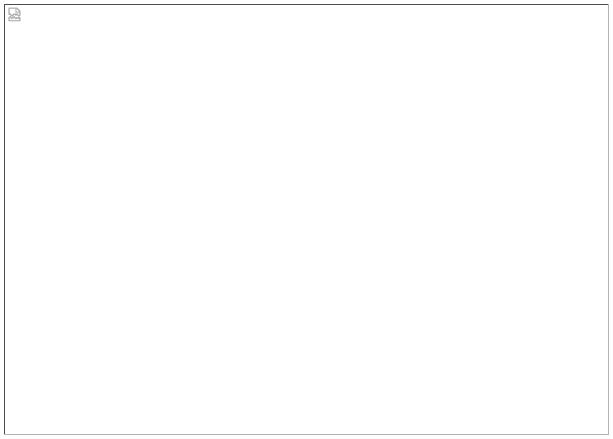


image.png1779x1266 81.5 KB

Now we can use Administrator account to obtain the access token.

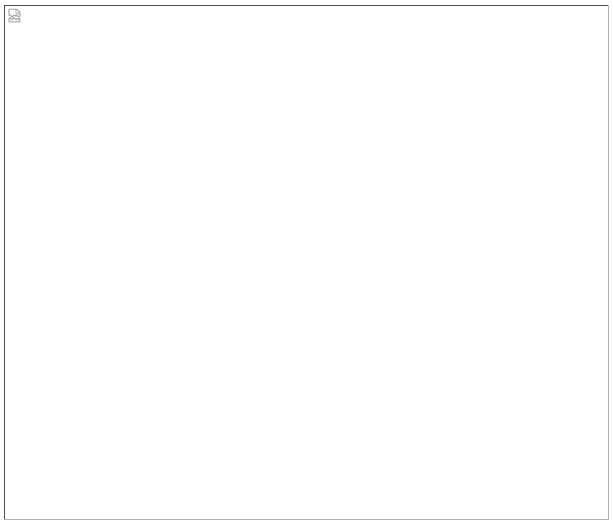


image.png1716x1460 228 KB

While posting my request I need to post highlighted parameters as `Form Data` format.

> I updated PM-POS to work with new Authentication features. You can check PM-POS source code to see details about how to format these requests. Specifically you'll find them in queries.js file. https://github.com/sambapos/pmpos/blob/master/app/queries.js#L59

However I received 400 response. Let's see why.

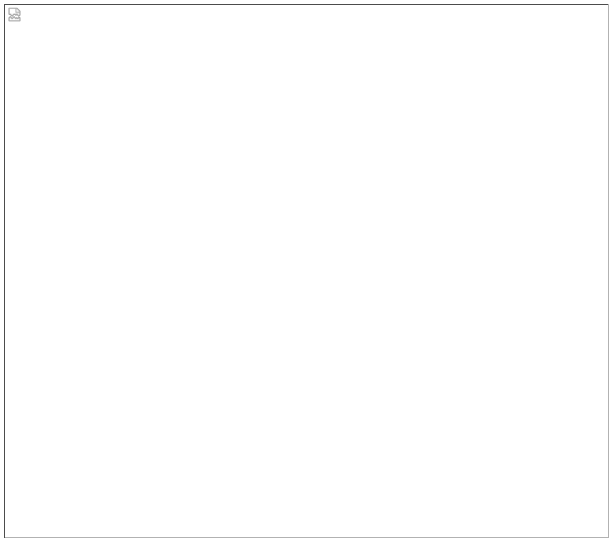


image.png1720x1520 169 KB

Our server says pmpos is not registered. You'll notice in pur post data we posted "pmpos" as clientid. For all authentication requests we need to tell the id of the client.

**So what is client id?**

We'll define a new Application for PM-POS in SambaPOS. You can find it under Users > Applications.

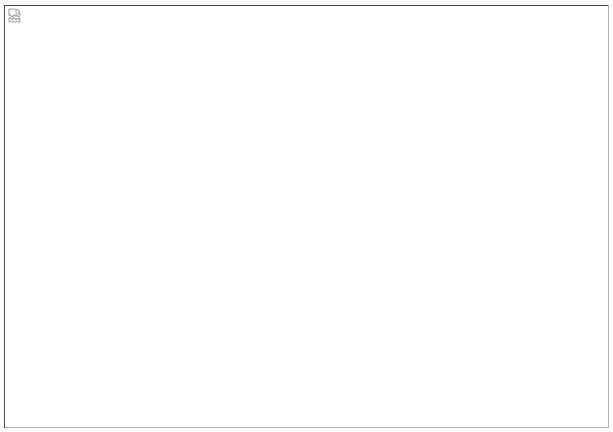


image.png1672x1171 132 KB

By defining the `pmpos` application you can configure how graphql server will respond to requests coming from pmpos application. When PM-POS application defined GraphQL application will respond to access token requests coming from pmpos client.

I'll post the request again.

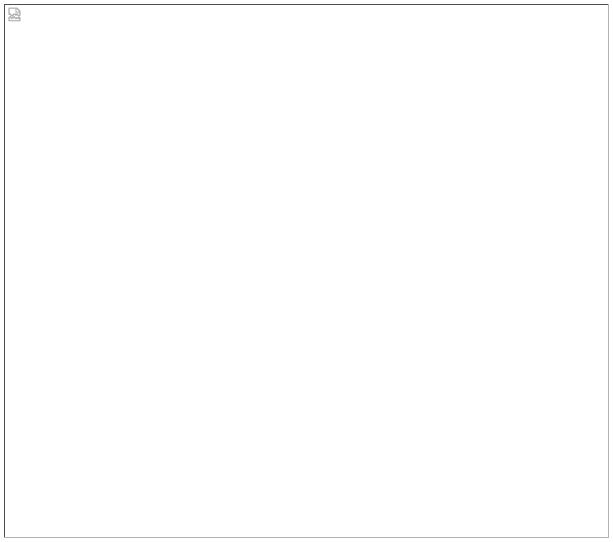


image.png1720x1520 192 KB

Now we received the access token for pmpos app.

> Does it still says "Client not registered"? If it says so that means GraphQL Server and SambaPOS connects to different databases. If you already setup SQL Server Express with multi terminal access you won't have that issue but if you're testing it with default `LocalDB` database server application probably connects to wrong database. On windows all services needs to logon with a windows user. By default MessagingServer service installs for Local System account. As it is a different user from your current windows user LocalDB creates a separate database for message server application. Either setup service to logon with the windows user account you're currently using or setup SQL Server Express.

With that access token we can execute graphql queries. My screenshot shows server responds fine to my GraphQL queries.

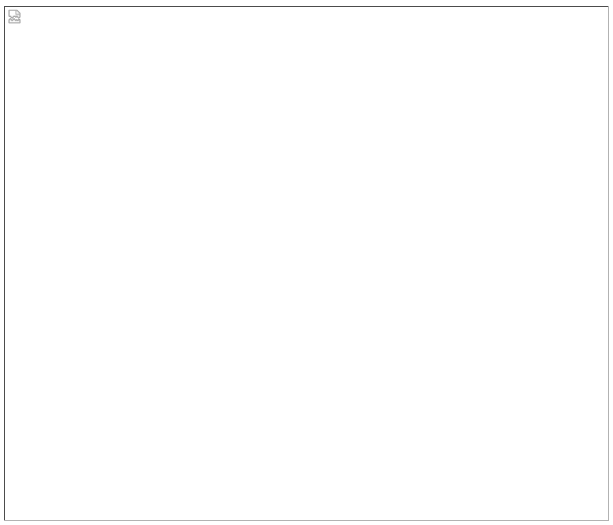


image.png1716x1460 259 KB

You can use that access token unlimited times without needing to use a user name and password again. However the lifetime for the access token is short and it will expire after a certain time.

**What happens when access token expires?**

When access token expires GraphQL server will start responding with **401 Unauthorized** Error. You need to obtain a new access token after it expires. So we'll need to use user name and password to be able to request a new access token.

This is what we don't want to do because we don't want user to repeatedly enter the user name and password so we consider hardcoding user name and password in source code. This is not acceptable for Javascript apps as source code can easily seen.

**This is where Refresh Tokens come into play…**

Instead of using user name or password we'll use Refresh Tokens to obtain a new access token. Maybe you already noticed when we first used the `user name` and `password` GraphQL server also gave us a Refresh Token.

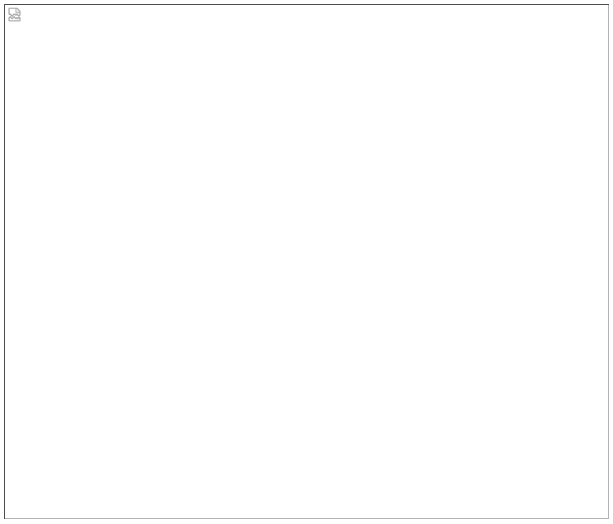


image.png1716x1460 187 KB

We can store that key in browser cache and when access token expires we can use it to obtain a new access token by sending a request to http://server:9000/Token URL.

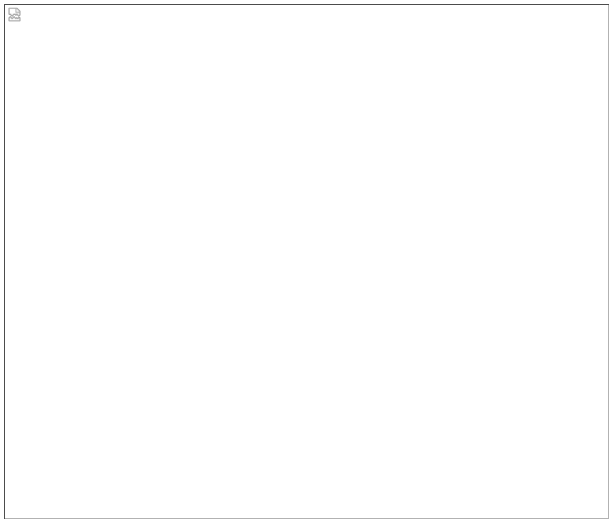


image.png1716x1460 240 KB

The refresh token can only be used once and when used server gives us another one.

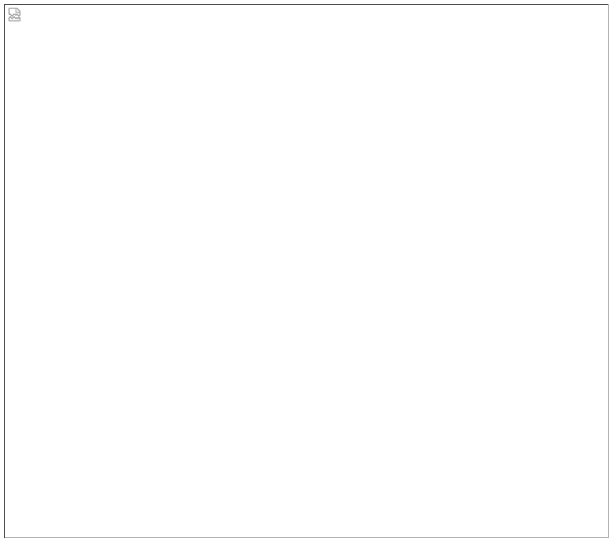


image.png1720x1520 195 KB

Refresh Tokens have really long lifetime. You'll remember we set Lifetime parameter as 365 days. That means pmpos client can receive a new access token with that refresh token in 365 days. (Unless we revoke it). So if that refresh token does not get lost an application can access graphql server without never needing to use the user name and password.

So the idea is.

1. We'll authenticate the app once with password and receive the `access token` and the `refresh token`.
2. Use `access token` until it expires and once expired we'll use refresh token to obtain a new `access token` and a `refresh token`.

You can see a list of active refresh tokens here.

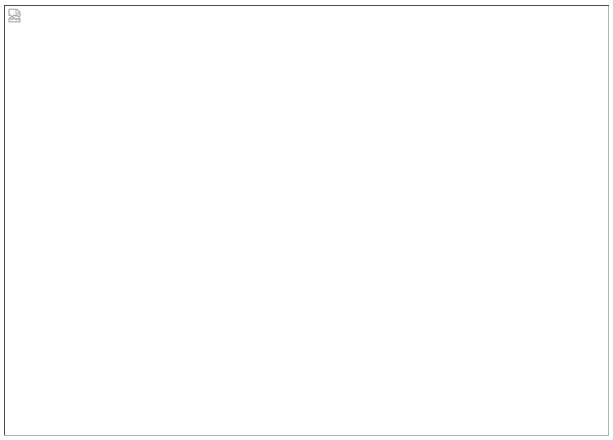


image.png1779x1266 92.2 KB

You can revoke a refresh token to disable it and force the client to re-login after access token expires.

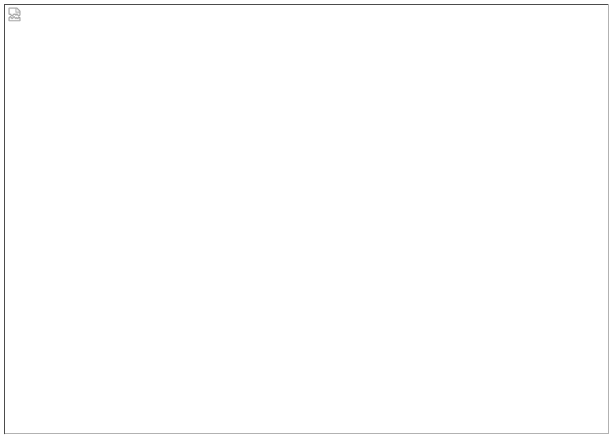


image.png1779x1266 87 KB

### Using device_id

If your application uses single account across multiple devices, authenticating on a device will revoke refresh tokens stored on other devices. To prevent this you can use device_id parameter to generate a separate refresh token per device.

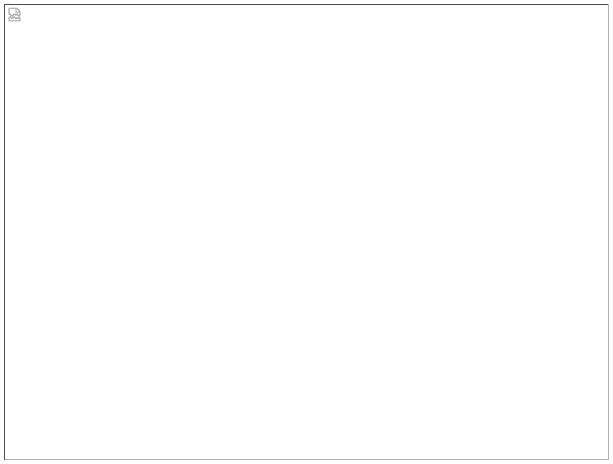


image.png1862x1405 89.3 KB

For native apps you can generate device_id from device. For browser apps you can generate a random number and store it in browser cache.

### Authorization

There are 3 Authorization options that can be set for the client.

- **All Functions on Local Network:** Client can execute all graphql functions if it makes requests in the local network.
- **All Functions:** There is no restriction for the client.
- **Permitted Functions:** From Permissions section you can choose which functions client can execute.

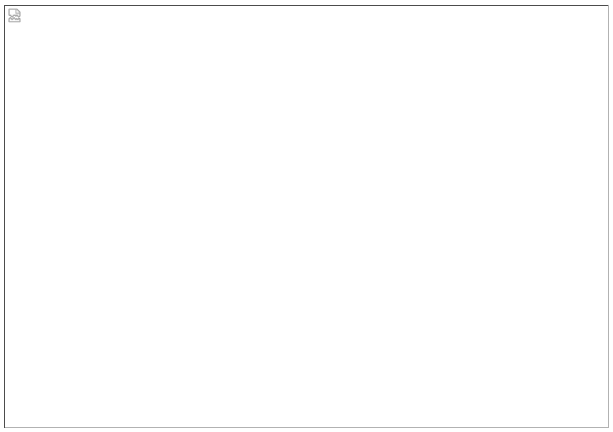


image.png1672x1171 161 KB

### Testing Authentication with PostMan

Before implementing GraphQL authentication into your application you may want to test it externally. I'll show you how you can test that by using Chrome's PostMan extension.

chrome.google.com

**Postman**

Postman makes API development faster, easier, and better. The free app is used by more than 3.5 million developers and 30,000…

I already have a user called **Administrator** and the password is **1111**
I also configured an application with **pmpos** key.

Now I'll post an authorization request via Postman.

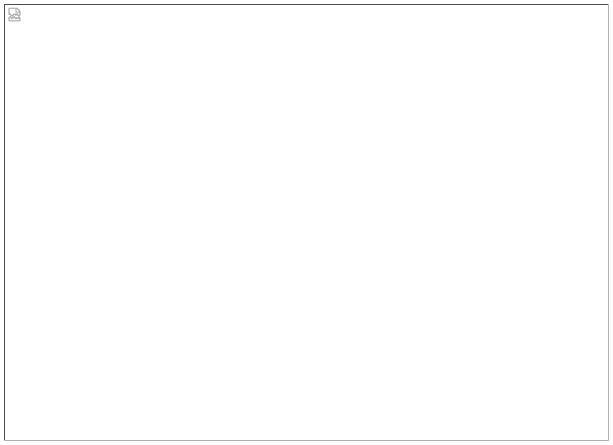


image.png1862x1346 185 KB

I copied access key and used it to execute a graphql.

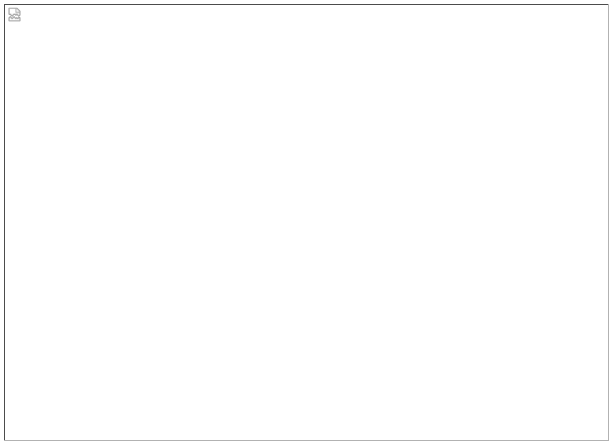


image.png1862x1346 124 KB

This is how Body tab appears.

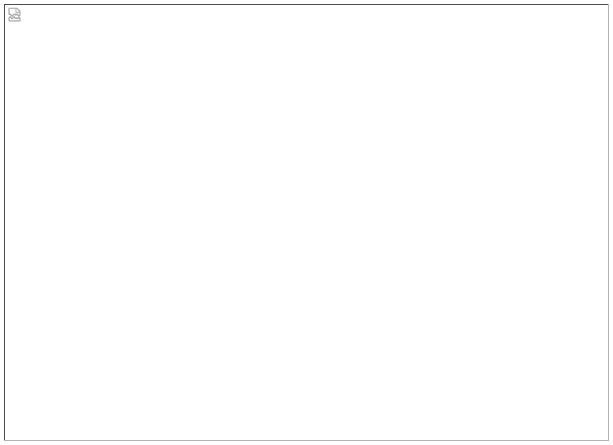


image.png1862x1346 87 KB

```
{query: "{getTickets{id}}", variables: null, operationName: null}
```

Now I'll use refresh token to obtain a new access token.

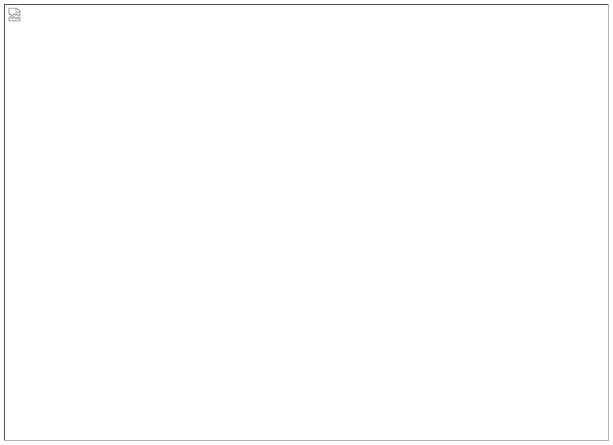


image.png1862x1346 185 KB

You can also use Postman to test graphql api when you need to quickly execute a query.

---

**emre** 2017-02-15 11:43:11 UTC #4

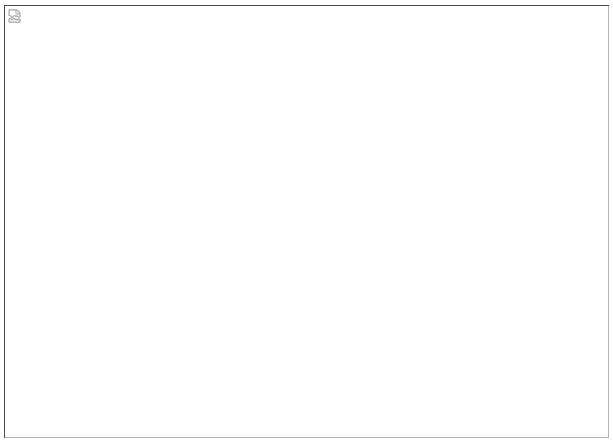
## GraphiQL

GraphiQL is a test application that helps us to test our GraphQL scripts and see the documentation for it.

**TIP**: notice the **i** in Graph**i**QL. GraphQL (no **i**) is the *language* we use, while Graph**i**QL (with **i**) is the **Interface** to the GraphQL Engine where we can test GraphQL queries and mutations and see the GraphQL documentation for all supported queries and mutations.

Navigate to http://server:9000 URL with your web browser. GraphiQL application should appear.

> You'll use your server url here. For example If your server ip is **192.168.1.4** and message server configured to use **8080** port you'll use http://192.168.1.4:8080 URL. Or if your server's network name is **ourserver** and message server configured to use **9090** port, the URL will be http://ourserver:9090
>
> If it does not work your server may not configured to accept remote connections. If you can run it on Server machine by navigating to http://localhost:9000 URL you'll need to enable remote access by running service with a user that have Administrator privileges. Review Server Setup section for more info.



Pasted image1681x1205 88.8 KB

When you first run it you'll receive an error message like this.

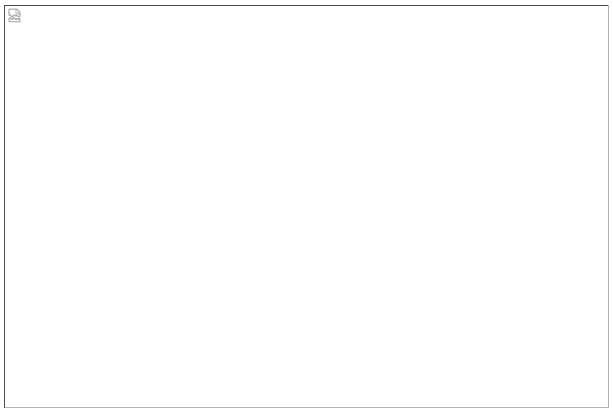


image.png1636x1092 69.4 KB

This is because a client access for **graphiql** client is not allowed. Follow these steps to enable graphiql application.

### Step 1

On SambaPOS > Management screen navigate to Users > Applications and create an application for **graphiql** client.

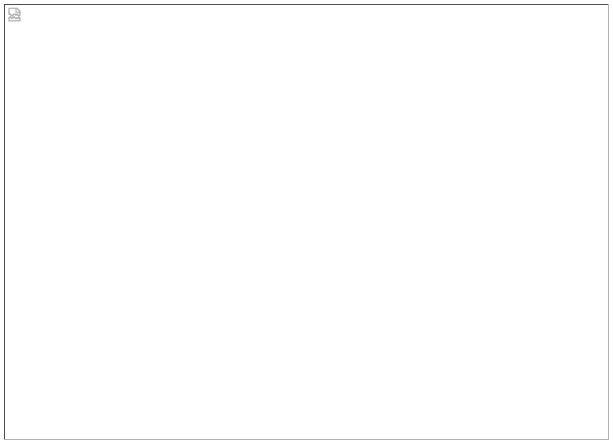


image.png1642x1184 95.4 KB

**Step 2**

With your web browser navigate to http://server:9000/login URL and enter the user name and password.

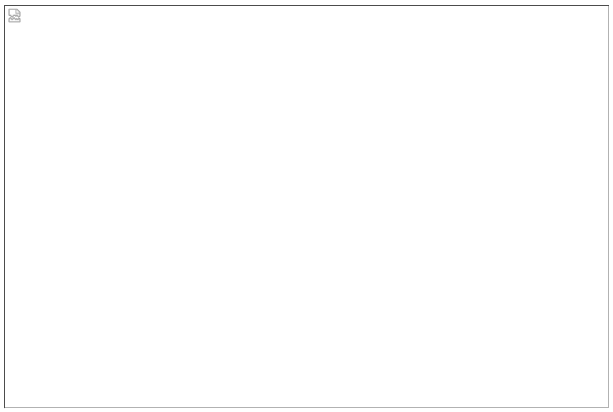


image.jpg1636x1092 90.5 KB

Now you can execute GraphQL queries.

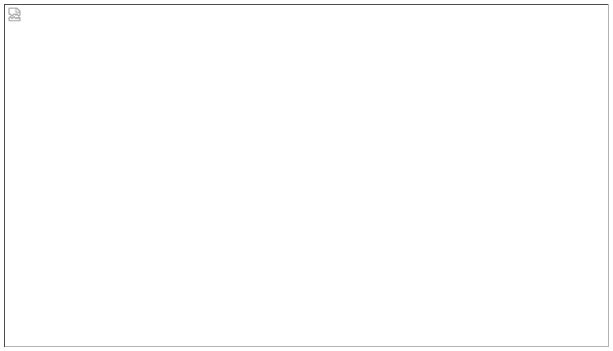


image.png1616x920 53.5 KB

---

**emre** 2017-02-15 12:03:53 UTC #5

## Using GraphQL API

GraphQL is an API language to allow external applications to access SambaPOS features.

### GraphQL Query structure

GraphQL queries formatted as a JSON like structure. There are two basic terms to remember. A `Query` and a `Mutation`. We'll use queries to query data and mutations to mutate (change) data.

### Queries

A query have a query name and a result format.

```
{getGlobalSettings{name,value}}
```

This will execute `getGlobalSettings` query and expects result as name & value format.

If we have 2 global settings stored in database query should return a JSON formatted result.

```
{
   "data":{
      "getGlobalSettings":[
         {
            "name":"Promotion",
            "value":"Active"
         },
         {
            "name":"Printing",
            "value":"Active"
         }
      ]
   }
}
```

We have two settings named as `Promotion` and `Printing`.

# Aliases

You'll notice query result includes function names and field names as we define them. You can optionally use aliases to change names defined in queries.

```
{settingNames:getGlobalSettings{setting:name}}
```

`settingNames` is an alias for function name name and we want name field named as `setting`. This is how result appears. In this query we only want names so values does not appear in result.

```
{
   "data":{
      "settingNames":[
         {
            "setting":"Promotion"
         },
         {
            "setting":"Printing"
         }
      ]
   }
}
```

### Arguments

To query a single setting we'll use `getGlobalSetting` function. To be able to query a single setting by name we'll use arguments. We can define an argument by typing it after function name in parenthesis.

```
{getGlobalSetting(name:Promotion){name,value}}
```

We're executing `getGlobalSetting` function with `name` argument. So the result will display setting named as `Promotion`.

```
{
   "data":{
      "getGlobalSetting":{
         "name":"Promotion",
         "value":"Active"
      }
   }
}
```

You'll notice result contains single result instead of an array on previous example.

This is what we should see when we query a setting named `NotExists`.

```
{
   "data":{
      "getGlobalSetting":{
         "name":"NotExists",
         "value":null
      }
   }
}
```

### Mutations

Until now we used shortcut syntax for queries. Full syntax for a query starts with `query` keyword and a custom name for the query. For example it may start with `query myQuery`.

```
query myQuery{getGlobalSettings{name}}
```

Naming queries is useful while using some advanced techniques to merge results of multiple queries but we won't cover that for now. We should remember for queries we can skip query keyword and query name.

Mutation is a special kind of query that we use to mutate (change) data. For example to update or delete a global setting we need to use a mutation. Unlike queries we should start it with mutation keyword and give it a name. There is no shortcut syntax for that.

```
mutation myMutation{
    updateGlobalSetting(name:Promotion,value:Disabled){name,value}
}
```

This mutation calls updateGlobalSetting function with two arguments. Name and value and changes Promotion setting as Disabled.

```
updateGlobalSetting(name:Promotion,value:Disabled)
```

Like queries all mutations should return a result so we also append result query format to the mutation.

```
{name,value}
```

So mutation will return resulting name and value.

This is how the mutation should respond after a successful call.

```
{
   "data":{
      "updateGlobalSetting":{
         "name":"Promotion",
         "value":"Disabled"
      }
   }
}
```

Another mutation we have can be used to delete settings from database.

```
mutation m{deleteSetting(name:Promotion){name,value}}
```

That should also return deleted value as the query result.

```
{
   "data":{
      "deleteSetting":{
         "name":"Promotion",
         "value":"Disabled"
      }
   }
}
```

### Accessing API documentation

By clicking `Docs` button on GraphiQL app you can browse API documentation.

Pasted image2166x1264 129 KB

## Testing what we learned so far

**Get a list of global settings**

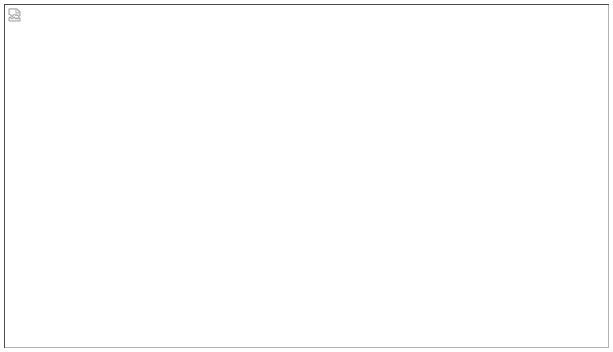


image.png1616x920 31.2 KB

We have no global settings atm.

**Set / Create a Global Setting**

image.png1616x920 42.8 KB

```
mutation m {
  updateGlobalSetting(name: "Integration", value: "Active") {
    name
    value
  }
}
```

This query sets **Integration** setting as **Active**. As you've set in your query SambaPOS responds with the name and value of new setting.

We'll execute `{getGlobalSettings{name,value}}` query once more to see if our setting is stored fine or not.

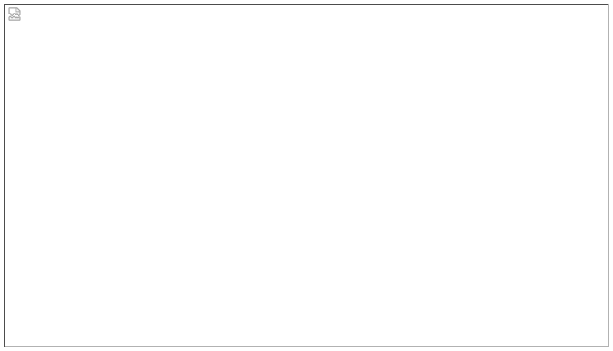


image.png1616x920 36 KB

That value is stored in database so you can use these functions to keep some setting like data in SambaPOS database.

When needed you can use documentation section to see functions relates with your query.

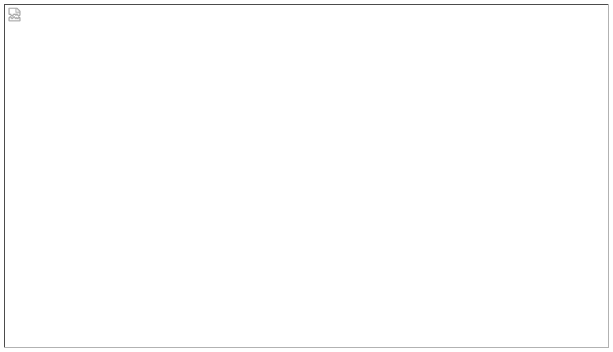


image.png1616x920 66.7 KB

---

**emre** 2017-02-15 12:33:40 UTC #6

## Writing a Query to Fetch a SambaPOS Menu

**Important Hint:** Message Server heavily caches some configuration data so *recent changes on configuration data may not appear on message server immediately*.

When you change something (for example change the price of a menu item) on SambaPOS > Management section please click [Main Menu] button to exit from Management Screen. That will reset Server's cache. Alternatively you can execute `mutation m{postResetCacheMessage{id}}` function to trigger reset cache on all SambaPOS Terminals.

Here you can see my flow to write & test a GraphQL query that fetches a menu from SambaPOS.

2017-02-15_15-24-58.gif840x464 531 KB

By using API documentation as a reference writing queries will be easier!

You'll form each query for your needs. You may need only category names or all product names for each category. For example if your products have multiple portions you can fetch a list of portions instead of just fetching the price of default portion.

```
{
  getMenu(name: "Menu") {
    categories {
      name
      menuItems {
        name
        product {
          portions {
            name
            price
          }
        }
      }
    }
  }
}
```

We can see Coffee Product have 3 portions.

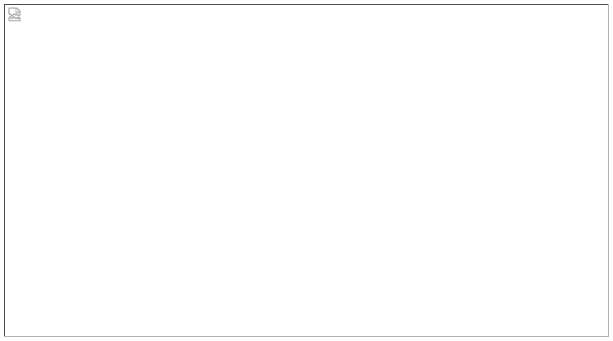


image.png841x464 32.9 KB

PS: If your application fetches menus for order taking you may consider to create a separate menu specific for your application in SambaPOS. By this way users may have different menu setups for your application.

## Fetching Order Tags (Modifiers)

For order taking applications you need to know what modifiers are configured for a product. For example when a Muffin is added a topping for Muffin might be asked. We can use `getOrderTagGroups` query for that.

image.png933x491 36.1 KB

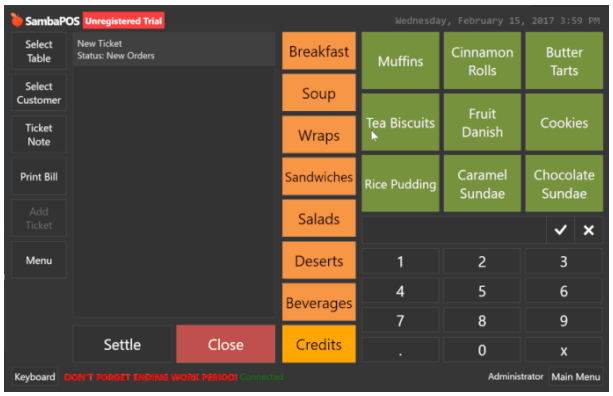
Here we can see the topping options for default portion of Muffin. As min,max parameters suggests operator have to select one modifier for Muffins.

Order Tags might be mapped to portions as restaurants may offer different choices for Muffins. To ensure you're fetching order tags mapped to portions you can include portion parameter.

image.png933x491 27.5 KB

For 2 Toppings portion operator required to make 2 selections.

Here you can see how it works in SambaPOS.




2017-02-15_15-59-48.gif930x594 464 KB

When I select 2 Toppings portion I can select 2 Toppings. When I select 3 Toppings portion price is different and I can make 3 Topping selections.

---

**emre** 2017-02-15 14:19:38 UTC #7

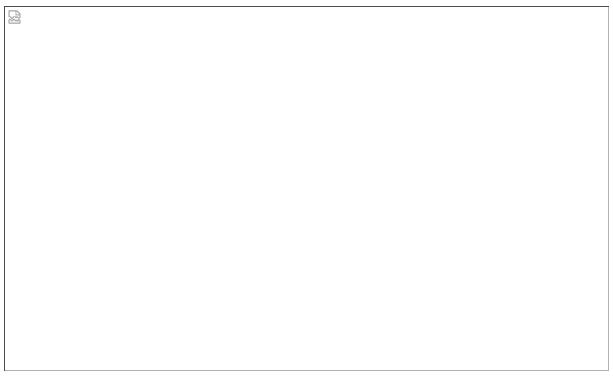
## Creating Tickets

There are two ways to create a ticket.

### Simple way

We can use `addTicket` mutation to create Tickets.

On this sample executing each query creates a separate ticket.



2017-02-15_16-40-47.gif798x482 275 KB

```
mutation m {
  addTicket(ticket: {orders: [{name: "Tea"},{name:"Milk"}]}) {
```

```
    totalAmount
  }
}
```

This mutation adds two orders and wants totalAmount as the result.

If you want to assign ticket to a Table you can use Entities Field.

```
mutation m {
  addTicket(ticket: {
    orders: [{name: "Tea"}, {name: "Milk"}],
    entities: [{entityType: "Tables", name: "B10"}]}) {
    totalAmount
  }
}
```

By using API fields like quantity, price, etc you can change how orders are added.

However these mutations will not execute certain Ticket UI Events like “Ticket Displayed, Before Ticket Close, Ticket Closed, etc.” So you need to handle how entity states updates or tickets prints.

### Enabling automation

if you want SambaPOS to execute Automation Rules while creating a ticket you need to use **Terminal API** instead.

### 1. Registering A Terminal

This step starts a session on SambaPOS server and it will work like a regular SambaPOS client. For this reason you need to define some parameters while registering a terminal.

image.png798x483 18 KB

```
mutation m{registerTerminal(
  ticketType:"Ticket"
  terminal:"Server"
  department:"Restaurant"
  user:"Admin"
)}
```

So this ticket will work like a ticket that have a “Ticket” ticket type, created on “Restaurant” department through “Server” terminal by user “Admin”.

This mutation will respond with a terminal Id like:

```
3K70v9cYZ02XJ7M-SC5LFw
```

Instead of re-sending user, ticket type, etc, parameters we’ll use this ID while executing next mutations.

### 2. Starting a ticket

On this step we’ll execute createTerminalTicket mutation to start a new ticket.

> On this step you can also open an existing ticket to add new orders.

image.png798x483 27.8 KB

When we execute second query server responds us with the unique id of the Ticket. That means ticket created successfully and we can add orders.

### 3.Adding Order

We’ll use `addOrderToTerminalTicket` to add orders.

image.png798x483 28.6 KB

```
mutation m3{addOrderToTerminalTicket(
  terminalId:"3K70v9cYZ02XJ7M-SC5LFw"
  productName:"Milk")
  {totalAmount}
}
```

This mutation adds Milk order in the ticket. It is important to remember each mutation will execute related SambaPOS rules if any. So if SambaPOS is configured to add “Free Cookies” product when a Milk is added your mutation will also trigger that rule.

#### 4. Closing Ticket

Until this step ticket stays in Server Memory in draft state. On this step we’ll close ticket to persist it to database and execute related “Ticket Closing” rules. For example if tickets auto prints on ticket close it will trigger ticket prints.

image.png798x483 60.6 KB

Now our ticket appears fine in SambaPOS.

> You’ll notice User appears as * in SambaPOS. I intentionally set user name as Admin instead of Administrator. If you see Users named as * you can check RegisterTerminal mutation to ensure user name set correctly.

I executed this mutation to close the ticket.

```
mutation m4{closeTerminalTicket(
  terminalId:"3K70v9cYZ02XJ7M-SC5LFw"
)}
```

#### 5. Unregistering Terminal

If we won’t create additional tickets we can unregister terminal to close our session on SambaPOS side. If you keep a session idle too long it may expire.

image.png798x483 36 KB

```
mutation m5{unregisterTerminal(
  terminalId:"3K70v9cYZ02XJ7M-SC5LFw"
)}
```

#### More Info

This is what we need to do to create tickets as if it created through a SambaPOS terminal. You can use API documentation for more info about api functions you can use while creating tickets.

Here you can find all mutations mentioned on this section.

```
mutation m{registerTerminal(
  ticketType:"Ticket"
  terminal:"Server"
  department:"Restaurant"
  user:"Admin"
)}

mutation m2{createTerminalTicket(
  terminalId:"3K70v9cYZ02XJ7M-SC5LFw")
  {uid}
}
```

```
mutation m3{addOrderToTerminalTicket(
  terminalId:"3K70v9cYZ02XJ7M-SC5LFw"
  productName:"Milk")
  {totalAmount}
}

mutation m4{closeTerminalTicket(
  terminalId:"3K70v9cYZ02XJ7M-SC5LFw"
)}

mutation m5{unregisterTerminal(
  terminalId:"3K70v9cYZ02XJ7M-SC5LFw"
)}
```

---

**emre** 2017-02-24 13:15:28 UTC #8

## Sample Integration

Here is a sample integration to integrate Gloria Food online ordering system to SambaPOS

### Gloria Food Setup

You'll need to obtain an API key from gloriafood.com.

image.jpg2560x1388 484 KB

You'll use this key while setting up integration application so you can copy it somewhere.

### SambaPOS Setup

image.png1073x650 59 KB

1. Sample is based on our Delivery Setup Tutorial. I added an additional ticket lister widget to display tickets which Delivery State is `Unconfirmed`. When a customer orders food for the first time it creates ticket as Unconfirmed so you can call customer and confirm their address. After confirmation further tickets will appear under `Waiting Orders` section. It prints tickets to kitchen for confirmed customers immediately. Unconfirmed tickets will print to kitchen on confirmation.
2. You'll need to create an order tag group called `Default`. Modifiers will apply to that group as free tags. If you want to use detailed groups you can map them inside source code.
3. You need to have a custom product tag called `Gloria Name` and type product names as they appear on Gloria Food there. SambaPOS will match products by using that custom tag.
4. You'll setup Phone field as Customer Entity's primary field. We check customer's existence by searching with phone number.
5. You'll add First Name, Last Name and EMail custom entity fields for customer entity type and setup entity type's display format as `[First Name] [Last Name]`
6. `Confirm` Command loads ticket, updates ticket state as Waiting and updates entity's (Customer) CStatus as Confirmed. Finally it closes ticket.

To create this setup Automatically you can use Advanced Delivery Screen configuration task.

image.png1073x650 44.5 KB

### GraphQL Server Setup

To give access Integration Application to our GraphQL server we need to create a client configuration.

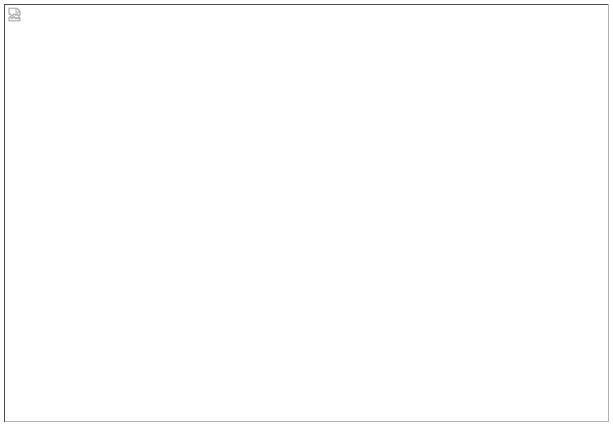


image.png1964x1360 120 KB

As integration application works on server side we can setup Authentication Type as Secret key and let application to authenticate with secret key instead of a user name and password. To create a secret key click `Generate Secret` button and type your secret key in the dialog. You need to type this key in integration application as explained on next section.

### Integration Application Setup

We need an application to read new orders from Gloria Food and creates tickets on SambaPOS side. This is a NodeJS application so you need to download and install NodeJS on your server. https://nodejs.org

After setting up NodeJS:

1. Create package.json and script.js files on a folder.
2. Under that folder run `npm install` command to install dependency packages
3. Edit `gloriaFoodKey` variable to setup Gloria Food key.
4. Edit `serverKey` variable to setup GraphQL authentication key.
5. run `node script.js` command to start the application.

**package.json**

```
{
  "name": "gloria",
  "version": "1.0.0",
  "description": "",
  "main": "script.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.14.1",
    "querystring": "^0.2.0",
    "request": "^2.79.0"
  }
}
```

**script.js**

```
var express = require('express');
var request = require('request');
var querystring = require('querystring');
var app = express();

var messageServer = 'localhost';
var messageServerPort = 9000;
var gloriaFoodKey = 'KEY_HERE';
var serverKey = 'KEY_HERE';
var timeout = 30000;
var customerEntityType = 'Customers';
var itemTagName = 'Gloria Name';
var ticketType = 'Delivery Ticket';
var departmentName = 'Restaurant';
var userName = 'Administrator';
var terminalName = 'Server';
var printJobName = 'Print Orders to Kitchen Printer';
var additionalPrintJobs = [];  // array of additional print job names
var miscProductName = 'Misc';
var deliveryFeeCalculation = 'Delivery Service';
var tipCalculation = 'Tip';
var accessToken = undefined;
var accessTokenExpires = '';

function Authorize(callback) {
    accessToken = undefined;
    var form = { grant_type: 'client_credentials', client_secret: serverKey, client_id: 'gloria' };
    var formData = querystring.stringify(form);
    var contentLength = formData.length;

    request({
        headers: {
            'Content-Length': contentLength,
            'Content-Type': 'application/x-www-form-urlencoded'
        },
        uri: 'http://' + messageServer + ':' + messageServerPort + '/Token',
        body: formData,
        method: 'POST'
    }, function (err, res, body) {
        if (err) {
            console.log('Error while trying to authorize >', err.message);
            if (callback) callback();
        }
        else if (res.statusCode === 400) {
            console.log(body);
            if (callback) callback();
        }
        else {
            var result = JSON.parse(body);
            accessToken = result.access_token;
            accessTokenExpires = new Date(result['.expires']);
            if (callback) callback();
        }
    });
}

function gql(query, callback) {
    if (!accessToken) {
        console.log('Valid access Token is needed to execute GQL calls.')
        return;
    }
    var data = JSON.stringify({ query: query });
    request({
        headers: {
            'Content-Type': 'application/json',
            'Accept': 'application/json',
            'Authorization': 'Bearer ' + accessToken
        },
```

```
        uri: 'http://' + messageServer + ':' + messageServerPort + '/api/graphql',
        body: data,
        method: 'POST'
    }, function (err, res, body) {
        if (res.statusCode === 401) {
            console.log('Should Authorize...');
            Authorize(() => gql(query, callback));
        }
        else {
            var data = JSON.parse(body).data;
            if (callback) callback(data);
        }
    });
}

function readTickets(callback) {
    request({
        method: 'POST',
        uri: 'https://pos.gloriafood.com/pos/order/pop',
        headers: {
            'Authorization': gloriaFoodKey,
            'Accept': 'application/json',
            'Glf-Api-Version': '2'
        }
    }, function (err, res, body) {
        if (err) {
            console.log(`problem with request: ${err.message}`);
        } else {
            console.log(body);
            callback(JSON.parse(body));
        }
    });
}

app.get('/', function (req, res) {
    res.send(`Welcome to Gloria Food integration application.
    <br/>
    <ul>
        <li><a href="/gqltest">Click here to retrieve product list</a></li>
        <li><a href="/test">Click here to create a test ticket</a></li>
    </ul>`);
});

app.get('/gqltest', function (req, res) {
    gql('{getProducts{id,name,price}}', (data) => {
        data.getProducts.forEach(x => res.write(`<div>${x.name} $${x.price}</div>`))
        res.end();
    });
});

app.get('/ctest', function (req, res) {
    loadCustomer({ phone: "222-344 1123" }, data => {
        res.send(JSON.stringify(data));
    })
});

app.get('/test', function (req, res) {
    processTickets(JSON.parse(getTestData()));
    res.send("Ticket Created! See log for details");
});

app.listen(3000, function () {
    console.log('Gloria Food integration app listening on port 3000!');
    Authorize(() => loop());
});

function loop() {
    if (!accessToken) {
        console.log('There is no valid access token. Skipping...')
        Authorize();
    }
    else if (accessTokenExpires < new Date()) {
        console.log('Access Token Expired. Reauthenticating...');
        Authorize(() => loop());
        return;
    }
    else {
        console.log('Reading Tickets...');
        readTickets((tickets) => processTickets(tickets));
    }
    setTimeout(loop, timeout);
}

function processTickets(tickets) {
    if (tickets.count == 0) return;
    tickets.orders.forEach((order) => processOrder(order));
}

function processOrder(order) {
    var customer = {
        firstName: order.client_first_name,
        lastName: order.client_last_name,
        email: order.client_email,
        phone: order.client_phone,
        address: order.client_address,
        newCustomer: false
    }
    loadCustomer(customer, customer => {
        var services = order.items
            .filter(x => x.type === 'tip' || x.type === 'delivery_fee')
            .map(x => { return { name: getCalculationName(x.type), amount: x.price }; })
            .filter(x => x.name);
        loadItems(order.items.map(x => processItem(x)), items => {
            createTicket(customer, items, order.instructions, order.fulfill_at, services, ticketId => {
                gql('mutation m {postTicketRefreshMessage(id:0){id}}', () => {
                    console.log(`Ticket ${ticketId} created...`);
                });
            });
        });
    });
}

function getCalculationName(name) {
    if (name === 'tip') return tipCalculation;
    if (name === 'delivery_fee') return deliveryFeeCalculation;
    return undefined;
}

function loadItems(items, callback) {
    var script = getLoadItemsScript(items);
    gql(script, data => {
        callback(items.filter(x => x.type === 'item').map(item => {
            return {
                id: item.id,
                name: item.name,
                type: item.type,
                sambaName: data[`i${item.id}`][0] ? data[`i${item.id}`][0].name : miscProductName,
                price: item.price,
                quantity: item.quantity,
                instructions: item.instructions,
                options: item.options
            }
        }));
    });
}

function isNewCustomer(customer) {
    if (customer.states && customer.states.find(x => x.stateName === 'CStatus')) {
        return customer.states.find(x => x.stateName === 'CStatus').state === 'Unconfirmed';
    }
    return false;
}
```

```
function createTicket(customer, items, instructions, fulfill_at, services, callback) {
    var newCustomer = isNewCustomer(customer);
    gql(getAddTicketScript(items, customer.name, newCustomer, instructions, fulfill_at, services), data => {
        if (newCustomer)
            callback(data.addTicket.id);
        else printTicketToKitchen(data.addTicket.id, () => callback(data.addTicket.id));
    });
}

function printTicketToKitchen(ticketId, callback) {
    gql(getKitchenPrintScript(ticketId), (data) => {
        if (additionalPrintJobs && additionalPrintJobs.length > 0) {
            var scripts = additionalPrintJobs.map((x) => getAdditionalPrintScript(x, ticketId)).join('\r\n');
            gql(scripts, callback);
        } else callback(data);
    });
}

function loadCustomer(customer, callback) {
    gql(getIsEntityExistsScript(customer), (data) => {
        if (!data.isEntityExists) {
            createCustomer(customer, callback);
        } else getCustomer(customer.phone, callback);
    });
}

function createCustomer(customer, callback) {
    gql(getAddCustomerScript(customer), (data) => {
        gql(getNewCustomerStateScript(customer), () => {
            getCustomer(data.addEntity.name, callback);
        })
    });
}

function getCustomer(customerName, callback) {
    gql(getCustomerScript(customerName), (data) => {
        callback(data.getEntity);
    });
}

function getLoadItemsScript(items) {
    var part = items.map(item => `i${item.id}: getProducts(itemTag:{name:"${itemTagName}",value:"${item.name}"}){name} `);
    return `{${part}}`;
}

function getCustomerScript(name) {
    return `{getEntity(type:"${customerEntityType}",name:"${name}"){name,customData{name,value},states{stateName,state}}}`;
}

function getIsEntityExistsScript(customer) {
    return `{isEntityExists(type:"${customerEntityType}",name:"${customer.phone}")}`;
}

function getAddCustomerScript(customer) {
    return `
    mutation m{addEntity(entity:{
        entityType:"${customerEntityType}",name:"${customer.phone}",customData:[
            {name:"First Name",value:"${customer.firstName}"},
            {name:"Last Name",value:"${customer.lastName}"},
            {name:"Address",value:"${customer.address}"},
            {name:"EMail",value:"${customer.email}"}
        ]})
        {name}
    }`;
}

function getNewCustomerStateScript(customer) {
    return `mutation m{updateEntityState(entityTypeName:"${customerEntityType}",entityName:"${customer.phone}",state:"Unconfirmed",stateName:"CStatus"){name}}`;
}

function getKitchenPrintScript(ticketId) {
    return `mutation m {
                executePrintJob(name: "${printJobName}", ticketId: ${ticketId},
                    orderStateFilters: [{stateName: "Status", state: "New"}],
                    nextOrderStates:[{stateName:"Status",currentState:"New",state:"Submitted"}])
                {name}
            }`;
}

function getAdditionalPrintScript(name, ticketId) {
    var mName = name.split(' ').join('_');
    return `mutation m_${mName} {
                n_${mName}:executePrintJob(name: "${name}", ticketId: ${ticketId})
                {name}
            }`;
}

function GetOrderTags(order) {
    if (order.options) {
        var options = order.options.map(x => `{tagName:"Default",tag:"${x.name}",price:${x.price},quantity:${x.quantity}}`);
        if (order.instructions && order.instructions !== '') {
            options.push(`{tagName:"Default",tag:"Instructions",note:"${order.instructions}"}`);
        }
        var result = options.join();
        return `tags:[${result}],`
    }
    return "";
}

function GetOrderPrice(order) {
    if (order.price > 0)
        return `price:${order.price},`;
    return "";
}

function getAddTicketScript(orders, customerName, newCustomer, instructions, fulfill_at, services) {
    var orderLines = orders.map(order => {
        return `{
            name:"${order.sambaName ? order.sambaName : order.name}",
            menuItemName:"${order.sambaName === miscProductName ? order.name : ''}",
            quantity:${order.quantity > 0 ? order.quantity : 1},
            ${GetOrderPrice(order)}
            ${GetOrderTags(order)}
            states:[
                {stateName:"Status",state:"New"}
            ]
        }`;
    });

    var entityPart = customerName
        ? `entities:[{entityType:"${customerEntityType}",name:"${customerName}"}],`
        : '';
    var calculationsPart = services
        ? `calculations:[${services.map(x => `{name:"${x.name}",amount:${x.amount}}`).join()}],`
        : '';

    var notePart = instructions && instructions !== ''
        ? `note:"${instructions}",`
        : '';
    var result = `
        mutation m{addTicket(
            ticket:{type:"${ticketType}",
                department:"${departmentName}",
                user:"${userName}",
                terminal:"${terminalName}",
                ${notePart}
                ${entityPart}
                states:[
                    {stateName:"Status",state:"Unpaid"},
                    {stateName:"Source",state:"Gloria"},
                    {stateName:"Delivery",state:"${newCustomer ? 'Unconfirmed' : 'Waiting'}"}
```

```
                ],
                tags:[{tagName:"Delivery Minutes",tag:"${Math.ceil(Math.abs(new Date(fulfill_at) - Date.now()) / 60000)}"}],
                ${calculationsPart}
                orders:[${orderLines.join()}]
            }){id}}`;
    return result;
}

function processItem(item) {
    var result = {
        id: item.id,
        name: item.name,
        type: item.type,
        price: item.price,
        quantity: item.quantity,
        instructions: item.instructions,
        options: item.options.map(x => { return { name: x.name, quantity: x.quantity, price: x.price } })
    };
    return result;
}

var getTestData = () => `{
    "count": 1,
    "orders": [
        {
            "coupons": [],
            "id": 776113,
            "restaurant_id": 4172,
            "client_id": 188995,
            "type": "delivery",
            "source": "website",
            "sub_total_price": 47.88,
            "tax_value": 4.13,
            "total_price": 62.41,
            "client_first_name": "John",
            "client_last_name": "Pink",
            "client_email": "john.brown@sambapos.com",
            "client_phone": "${Math.floor((Math.random() * 300) + 200)}-456 6699",
            "pin_skipped": 0,
            "restaurant_name": "John's Excellent Pizza",
            "restaurant_phone": "+15558964567",
            "restaurant_country": "United States of America",
            "restaurant_state": "California",
            "restaurant_city": "San Francisco",
            "restaurant_street": "10 Market Street",
            "restaurant_zipcode": "1234678",
            "restaurant_latitude": "37.7944872589999",
            "restaurant_longitude": "-122.395311999999",
            "instructions": "Deliver ASAP",
            "currency": "USD",
            "latitude": "37.79448725889753",
            "longitude": "-122.395311680426",
            "tax_type": "NET",
            "tax_name": "Sales Tax",
            "fulfill_at": "${ new Date(new Date().getTime() + 25 * 60000).toISOString()}",
            "pos_system_id": 1,
            "restaurant_key": "8yCPCvb3dDo1k",
            "api_version": 2,
            "payment": "ONLINE",
            "client_address": "21 Market Street, San Francisco",
            "items": [
                {
                    "id": 1678316,
                    "name": "DELIVERY_FEE",
                    "total_item_price": 5,
                    "price": 5,
                    "quantity": 1,
                    "instructions": null,
                    "type_id": null,
                    "type": "delivery_fee",
                    "tax_rate": 0.1,
                    "tax_value": 0.5,
                    "parent_id": null,
                    "cart_discount_rate": 0,
                    "cart_discount": 0,
                    "tax_type": "NET",
                    "item_discount": 0,
                    "options": []
                },
                {
                    "id": 1678317,
                    "name": "TIP",
                    "total_item_price": 5.67,
                    "price": 5.67,
                    "quantity": 1,
                    "instructions": null,
                    "type_id": null,
                    "type": "tip",
                    "tax_rate": 0.05,
                    "tax_value": 0.2702,
                    "parent_id": null,
                    "cart_discount_rate": 0,
                    "cart_discount": 0,
                    "tax_type": "GROSS",
                    "item_discount": 0,
                    "options": []
                },
                {
                    "id": 1678322,
                    "name": "Pizza Margherita",
                    "total_item_price": 8.2,
                    "price": 7,
                    "quantity": 1,
                    "instructions": "",
                    "type_id": 58424,
                    "type": "item",
                    "tax_rate": 0.07,
                    "tax_value": 0,
                    "parent_id": 1678332,
                    "cart_discount_rate": 0,
                    "cart_discount": 0,
                    "tax_type": "NET",
                    "item_discount": 8.2,
                    "options": [
                        {
                            "id": 1771325,
                            "name": "Small",
                            "price": 0,
                            "group_name": "Size",
                            "quantity": 1,
                            "type": "size"
                        },
                        {
                            "id": 1771326,
                            "name": "Crispy",
                            "price": 0,
                            "group_name": "Crust",
                            "quantity": 1,
                            "type": "option"
                        },
                        {
                            "id": 1771327,
                            "name": "Extra mozzarella",
                            "price": 1.2,
                            "group_name": "Extra Toppings (Small)",
                            "quantity": 1,
                            "type": "option"
                        }
                    ]
                },
                {
                    "id": 1678324,
```

```
    "name": "Pizza Prosciutto",
    "total_item_price": 11.7,
    "price": 8,
    "quantity": 1,
    "instructions": "User may enter a very long description for the pizza. For example he may want to explain what kind of sauce he wants or how dough should be cooked. So we should handle that case properly.",
    "type_id": 58425,
    "type": "item",
    "tax_rate": 0.07,
    "tax_value": 0.819,
    "parent_id": 1678332,
    "cart_discount_rate": 0,
    "cart_discount": 0,
    "tax_type": "NET",
    "item_discount": 0,
    "options": [
        {
            "id": 1771331,
            "name": "Large",
            "price": 2,
            "group_name": "Size",
            "quantity": 1,
            "type": "size"
        },
        {
            "id": 1771332,
            "name": "Crispy",
            "price": 0,
            "group_name": "Crust",
            "quantity": 1,
            "type": "option"
        },
        {
            "id": 1771333,
            "name": "Extra mozzarella",
            "price": 1.7,
            "group_name": "Extra Toppings (Large)",
            "quantity": 1,
            "type": "option"
        }
    ]
},
{
    "id": 1678331,
    "name": "Pizza Prosciutto",
    "total_item_price": 8.7,
    "price": 8,
    "quantity": 2,
    "instructions": "no salt",
    "type_id": 58425,
    "type": "item",
    "tax_rate": 0.07,
    "tax_value": 0.609,
    "parent_id": 1678332,
    "cart_discount_rate": 0,
    "cart_discount": 0,
    "tax_type": "NET",
    "item_discount": 0,
    "options": [
        {
            "id": 1771343,
            "name": "Small",
            "price": 0,
            "group_name": "Size",
            "quantity": 1,
            "type": "size"
        },
        {
            "id": 1771344,
            "name": "Fluffy",
            "price": 0,
            "group_name": "Crust",
            "quantity": 1,
            "type": "option"
        },
        {
            "id": 1771345,
            "name": "Corn",
            "price": 0.7,
            "group_name": "Extra Toppings (Small)",
            "quantity": 1,
            "type": "option"
        }
    ]
},
{
    "id": 1678332,
    "name": "2 + 1 Pizza Special",
    "total_item_price": 28.6,
    "price": 0,
    "quantity": 1,
    "instructions": null,
    "type_id": 251,
    "type": "promo_item",
    "tax_rate": 0.07,
    "tax_value": 1.3566,
    "parent_id": null,
    "cart_discount_rate": 0.05,
    "cart_discount": 1.02,
    "tax_type": "NET",
    "item_discount": 8.2,
    "options": []
},
{
    "id": 1678334,
    "name": "Spaghetti Bolognese",
    "total_item_price": 18,
    "price": 9,
    "quantity": 2,
    "instructions": "",
    "type_id": 58426,
    "type": "item",
    "tax_rate": 0.07,
    "tax_value": 1.197,
    "parent_id": null,
    "cart_discount_rate": 0.05,
    "cart_discount": 0.9,
    "tax_type": "NET",
    "item_discount": 0,
    "options": []
},
{
    "id": 1678335,
    "name": "Spaghetti Frutti di Mare",
    "total_item_price": 12,
    "price": 12,
    "quantity": 1,
    "instructions": "",
    "type_id": 58427,
    "type": "item",
    "tax_rate": 0.07,
    "tax_value": 0.798,
    "parent_id": null,
    "cart_discount_rate": 0.05,
    "cart_discount": 0.6,
    "tax_type": "NET",
    "item_discount": 0,
    "options": []
},
{
    "id": 1678336,
    "name": "5% off total larger than 40$",
    "total_item_price": 0,
```

```
                    "price": 0,
                    "quantity": 1,
                    "instructions": null,
                    "type_id": 250,
                    "type": "promo_cart",
                    "tax_rate": 0.07,
                    "tax_value": 0,
                    "parent_id": null,
                    "cart_discount_rate": 0.05,
                    "cart_discount": -2.52,
                    "tax_type": "NET",
                    "item_discount": 2.52,
                    "options": []
                }
            ]
        }
    ]
}`;
```

Gloria Food Integration Help

Task Editor - Separate KDS for Delivery

**emre** 2017-02-27 23:13:35 UTC #9

## Running Integration application as a Windows Service

Installing and running our Node.JS integration app as a windows service is really simple.

First of all you need to install Node-Windows npm package by using this command.

```
npm install -g node-windows
```

This command installs node-windows package globally on your PC. Next you'll run this command under the folder where you created script.js file.

```
npm link node-windows
```

After installation you'll create a service installation js file next to script.js file.

**service.js**

```
var Service = require('node-windows').Service;

var svc = new Service({
    name:'SambaPOS GloriaFood',
    description:'SambaPOS Integration Service for Gloria Food',
    script:require('path').join(__dirname,'script.js')
});

svc.on('install',function(){
  svc.start();
});

svc.on('uninstall',function(){
  console.log('Uninstall complete.');
  console.log('The service exists: ',svc.exists);
});


svc.install();
//svc.uninstall();
```

Finally you can execute this command to install the integration app as a service.

```
node service.js
```

The service should appear on services window.

image.png892x533 56 KB

**Hint:** Express Server module added to create test tickets by using the browser from http://localhost:4000 url. Before installing service for production you can remove test stuff from script so you won't unnecessarily publish a web service.

Here you can find the app that does not have test functions.
https://gist.github.com/emreeren/0d26aca9b83415fa01fcc3b5d2f172ad

## More Samples

I'm adding videos of some solutions that uses GraphQL API to integrate to SambaPOS. They are in Turkish language but will give an idea about possibilities. If you need more information feel free to contact us.

**SambaPOS GO Self Service Ordering**

**Android Client for Waiters**

**Device Integration**

We'll release more videos and sample source code. Let us know your ideas and suggestions.

**emre** 2017-03-16 13:33:55 UTC #10

**rightguys** 2017-03-17 18:14:49 UTC #11

Can I install the "Advanced Delivery Screen Setup" over the current one, even though I already have the "Custom Package Delivery for V5"?

**sukasem** 2017-03-17 19:19:53 UTC #12

I think you can just rename the existing one just in case.

**emre** 2017-03-17 19:22:31 UTC #13

S sukasem:

> I think you can just rename the existing one just in case.

Of course after backing up

R rightguys:

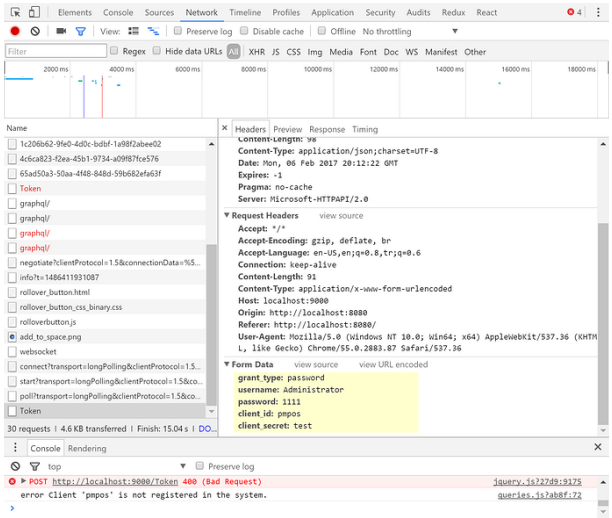
> Can I install the “Advanced Delivery Screen Setup” over the current one,

It is hard to guess without knowing your setup. Maybe it will give better idea if you can test that on a non production database.

---

**Manuel** 2017-03-17 22:34:27 UTC #14

hello @emre which program your using here?




i dont know where to write these commands…

thx for answers!

---

**markjw** 2017-03-17 23:18:32 UTC #15

M Manuel:

> which program your using here?

This is developer tools built into Google Chrome browser. Just open Chrome then from menu, More Tools > Developer Tools, or press Ctrl+Shift+I (letter i).

---

**kendash** 2017-03-17 23:54:51 UTC #16

@Manuel it would be best if you explained what you are trying to do. It looks like you really are not sure what you are doing but it is hard for us to help you because we really can not guess what your wanting.

---

**emre** 2017-03-18 00:28:37 UTC #17

markjw:

> Just open Chrome then from menu, More Tools > Developer Tools, or press Ctrl+Shift+I (letter i).

Wow didn’t know `CTRL + Shift + i` shortcut. I’m using `F12`.

---

**Manuel** 2017-03-18 00:30:34 UTC #18

thanks @markjw for the answer.

---

**Manuel** 2017-03-18 00:30:49 UTC #19

Hi @kendash im interesting too use this api integration Graph QL, my knowing about this is little small, but i want to try out… i want to use GloriaFood and this PMPOS it look realy nice…

my english is not the best i hope you understand.
I have a Restaurant in Paraguay.

btw. i do not know very much about this code writing…

---

**kendash** 2017-03-18 01:17:45 UTC #20

Ok well that information your referencing probably wont help you much. You should join the Beta team and look at some of our discussions on graphql there.

PMPOS project is here:

GitHub

**sambapos/pmpos**

pmpos - PM POS

---

**next page →**
Home Categories FAQ/Guidelines Terms of Service Privacy Policy