SALONHQ

Salon HQ Platform API Reference

Version 1.2.0

Prepared by: integrations@stepladder.co

SALONHQ

SALONHQ

Salon HQ Platform API Reference

Introduction

The SalonHQ API is organized around REST and uses standard HTTP response codes, authentication, and syntax.

The SalonHQ has three main endpoints:

1. **Products** - details about the items to be sold, including pricing (retail, wholesale, special), images, extended descriptions, inventory, and rules.
2. **Orders** - information about orders, including products purchased, quantity, unit pricing, total order amount, etc. Shipping and returns information can also be managed through this end point.
3. **Accounts** - if the partner system includes existing customer information (e.g., about Stylists), the Accounts endpoint can be used to communicate that data to/from SalonHQ.

Note: Both the Products and Orders end points contain a set of functions that can also be managed as components. For example, Inventory could be updated separately from Product Updates, and Shipping Status or Returns Notifications could be updated apart from the Order Updates.

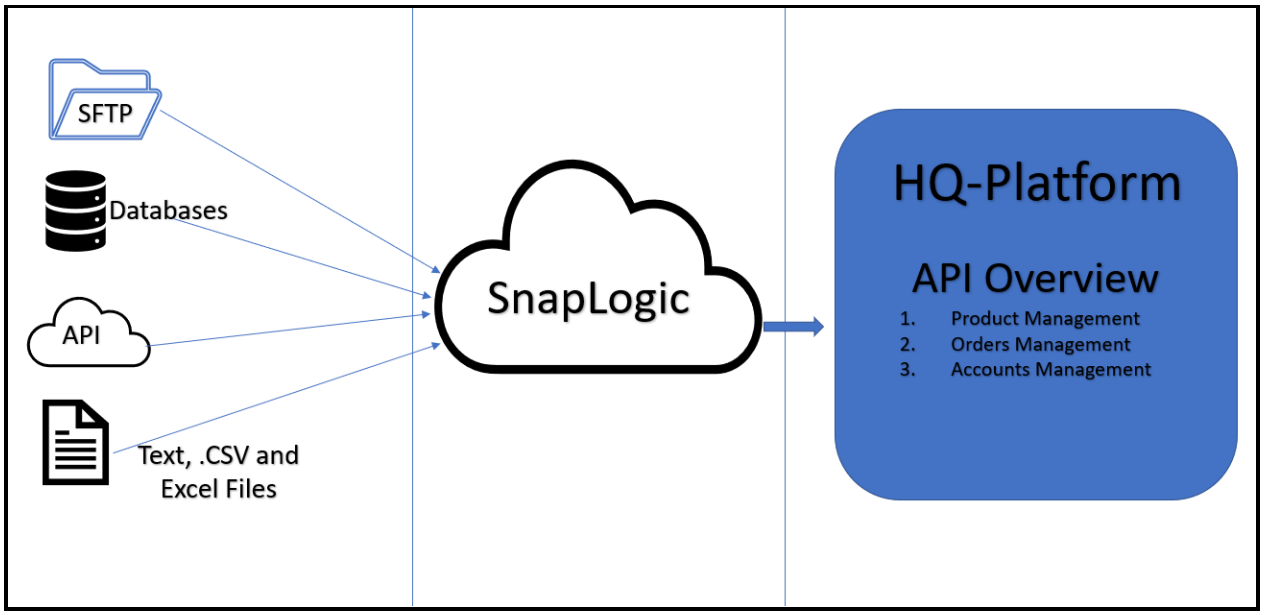
HQ-Platform Authentication

SalonHQ has both test and production instances. Presently, we are employing basic authentication for each instance. URLs and Credentials will be shared at the point of engagement.

Snaplogic Integration overview

Snaplogic is an iPaaS tool used for connecting cloud-based data sources, on-premises applications, files, and other data sources to the SalonHQ Platform.

It reduces the overall complexity of integrating with the SalonHQ Platform insomuch as it is a single point of interconnect for supplier systems and can simplify the details of interacting directly with the specific API endpoints themselves.



HQ-Platform provides many Application Programming Interfaces (APIs). These APIs offer a simple, powerful, and secure way of importing or exporting data to and from the HQ-Platform.

The data import operation loads data from source systems into HQ-Platform. The following are the primary integrations API's.

1. Product Management
2. Orders Management
3. Accounts Management

Product Management

With the Product API we have multiple subcategories to handle requirements related to product management.

All product related API's require a Supplier Id (Mandatory field) to get all available data related to:

a) Inventory
    a. Product ID
    b. Expected delivery date
    c. Product availability
b) Product descriptions
    a. Product ID
    b. SKU
    c. Name
    d. Description
c) Validation Rules
    a. Availability Rules
    b. Product sizes
d) Enhanced description
    a. Product enhanced description
e) Product images
    a. Image URL
f) Brand level details
    a. ID
    b. Name
g) Product Category
    a. ID
    b. Name
h) Price level details
    a. Currency code
    b. Currency Sign
    c. Product Type
    d. Price value - Retail
    e. Price value - Wholesale
    f. Price value - Promotion/Other

Below is a sample description of the PRODUCT endpoint. Once we pass supplier ID, it will provide all the data related Supplier with all properties and sub properties.

Pagination is available for this API.

The Payload of the Product object is detailed below. We have to pass supplier id to get all product level details. The response object will have all details by product.

Request

```
{
 -> supplier_uid* (String)
 -> page (Int)
 -> per_page (Int)
}
```

Response

```
{
 <- data: [
  <- uid (String)
  <- sku (String)
  <- name (String)
  <- description (String)
  <- active (Bool)
  <- image_url (String)
  <- availability: {
   <- availability_control (Bool)
   <- available_quantity (Int)
  }
  <- brand: {
   <- uid (String)
   <- name (String)
  }
  <- categories: [
   <- uid (String)
   <- name (String)
  ]
  <- prices: [
   <- currency_code (String) #USD
   <- currency_sign (String) #$
   <- type (String) #{regular, discounted, wholesale}
   <- value (Int) #value in cents
  ]
 ]
 <- page (Int)
 <- per_page (Int)
 <- total_pages (Int)
}
```

Payload for Product Creation: This API has the flexibility to support one row or a bulk load of data.

CreateProduct

Request

```
[
 -> supplier_uid* (String)
 -> data*: [
  -> sku* (String)
  -> name* (String)
  -> description (String)
  -> active (Bool)
  -> image_url (String)
  -> availability: {
   -> availability_control (Bool)
   -> available_quantity (Int)
  }
  -> brand_uid* (Sting)
  -> categories: [
   -> uid (String)
  ]
  -> prices*: [
   -> currency_code (String) #USD
   -> currency_sign (String) #$
   -> type* (String) #{regular, discounted, wholesale}
   -> value* (Int) #value in cents
  ]
 ]
]
```

Response

```
[
 <- uid (String)
 <- sku (String)
 <- name (String)
 <- description (String)
 <- active (Bool)
 <- image_url (String)
 <- availability: {
  <- availability_control (Bool)
  <- available_quantity (Int)
 }
 <- brand: {
  <- uid (String)
  <- name (String)
 }
 <- categories: [
  <- uid (String)
  <- name (String)
 ]
```

```
  <- prices: [
    <- currency_code (String) #USD
    <- currency_sign (String) #$
    <- type (String) #{regular, discounted, wholesale}
    <- value (Int) #value in cents
  ]
]
```

Payload for Product Update: This API has the flexibility to support one row or a bulk load of data.

UpdateProducts

Request

```
[
  -> supplier_uid* (String)
  -> data*: [
    -> uid* (String)
    -> sku (String)
    -> name (String)
    -> description (String)
    -> active (Bool)
    -> availability: {
      -> availability_control (Bool)
      -> available_quantity (Int)
    }
    -> brand_uid (Sting)
    -> categories: [
      -> uid (String)
    ]
    -> prices: [
      -> currency_code (String) #USD
      -> currency_sign (String) #$
      -> type (String) #{regular, discounted, wholesale}
      -> value (Int) #value in cents
    ]
  ]
]
```

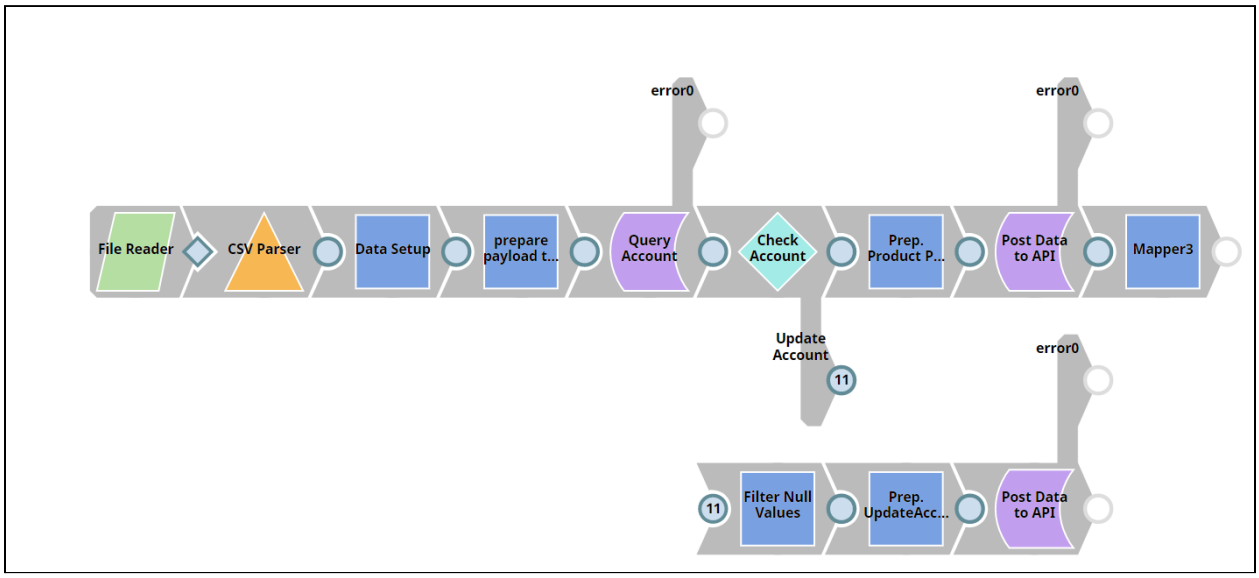
Response

```
[
  <- uid (String)
  <- sku (String)
  <- name (String)
  <- description (String)
  <- active (Bool)
```

```
 <- image_url (String)
 <- availability: {
   <- availability_control (Bool)
   <- available_quantity (Int)
 }
 <- brand: {
   <- uid (String)
   <- name (String)
 }
 <- categories: [
   <- uid (String)
   <- name (String)
 ]
 <- prices: [
   <- currency_code (String) #USD
   <- currency_sign (String) #$
   <- type (String) #{regular, discounted, wholesale}
   <- value (Int) #value in cents
 ]
]
```

Overview of a Snaplogic Process used for the product object.

Below is a sample process which reads data from a text file and process the data in API understandable formats. Once the data is formatted payload will be generated to Create/Update API of products. Snaplogic is capable of handling multiple formats from both web as well as on-premise systems.



Order Management

With the Orders API we have multiple subcategories to handle requirements related to order management.

All Order related API's we need to pass MarketPlaceId Id (Mandatory field) to get all available data related to.

a) Order Placement
    a. Order ID
    b. Product Details
        i. Product ID
        ii. Quantity
    c. Pricing details
b) Shipment Status
    a. Shipment Address
    b. Shipment tracking
    c. Delivery Date
c) Returns
    a. Product Details
    b. Reasons for return (Return Code)
    c. Date of return

Below is a description of the endpoint. Once we pass MarketPlaceId, it will provide all the data related orders and its related data and with all properties and sub properties. Pagination is available in this API and we need to process the data accordingly.

Payload for Order Management Create: This API has the flexibility to support one row or a bulk load of data.

Request

```
{
 -> marketplace_uid* (String)
 -> page (Int)
 -> per_page (Int)
}
```

Response

```
{
 <- data: [
  <- uid (String)
  <- created_at (String)
  <- number (String)
  <- buyer: {accountEntity}
```

```
   <- seller: {accountEntity}
   <- shipping_address: {addressEntity}
   <- order_type (String) #{regular, wholesale}
   <- shipping_status (String)
   <- payment_status (String)
   <- shipping (Int)
   <- tax (Int)
   <- tracking_url (String)
   <- order_items: [
     <- item: {
       <- uid (String)
       <- name (String)
       <- sku (String)
     }
     <- price (Int)
     <- qty (Int)
   ]
 ]
 <- page (Int)
 <- per_page (Int)
 <- total_pages (Int)
}
```

Payload for Order Management Update: This API has the flexibility to support one row or a bulk load of data.

Update Orders

Request

```
{
 -> marketplace_uid* (String)
 -> data: [
   -> order_uid* (String)
   -> shipping_status (String)
   -> tracking_url (String)
 ]
}
```

Response

```
[
 <- uid (String)
 <- created_at (String)
 <- number (String)
 <- buyer: {accountEntity}
```

```
 <- seller: {accountEntity}
 <- shipping_address: {addressEntity}
 <- order_type (String) #{regular, wholesale}
 <- shipping_status (String)
 <- payment_status (String)
 <- shipping (Int)
 <- tax (Int)
 <- tracking_url (String)
 <- order_items: [
  <- item: {
   <- uid (String)
   <- name (String)
   <- sku (String)
  }
  <- price (Int)
  <- qty (Int)
 ]
]
```

Account Management

With the Accounts API we have multiple subcategories to handle requirements related to Account management.

All Account related API's we need to pass MarketPlaceId Id (Mandatory field) to get all available data related to.

a) Customer Details
    a. Customer ID
    b. Contact Details
        i. Customer Name
        ii. Email
        iii. Phone
    c. Address
        i. Customer ID
        ii. Street 1
        iii. Street2
        iv. City
        v. Postal Code
        vi. State Code
        vii. Country Code
        viii. Type (Shipping/Billing)
    d. Order History

i. Order ID
ii. Product ID
iii. Date of Purchase
iv. Shipment Details

e. Returns History
i. Order ID
ii. Product ID
iii. Date of Return
iv. Return reason
v. Shipment Details

Below is the description of the endpoint. Once we pass MarketPlaceId, it will provide all the data related accounts and its related data and with all properties and sub properties. Pagination is available in this API and we need to process the data accordingly.

Payload for Account Management: This API has the flexibility of processing data by a row or in bulk.

Request

```
{
 -> marketplace_uid* (String)
 -> page (Int)
 -> per_page (Int)
}
```

Response

```
{
 <- data: [
  <- uid (String)
  <- contact: {
   <- formal_name (String)
   <- email (String)
   <- phone (String)
  }
  <- addresses: [
   <- uid (String)
   <- street_line_1 (String)
   <- street_line_2 (String)
   <- city (String)
   <- postal_code (String)
   <- state_code (String)
   <- country_code (String)
   <- type (String) #{shipping, billing}
  ]
  <- role (String)
 ]
```

```
 <- page (Int)
 <- per_page (Int)
 <- total_pages (Int)
}
```

Payload for Creating Accounts: This API has the flexibility of processing data by a row or in bulk.

Request

```
[
 -> marketplace_uid* (String)
 -> data*: [
  -> contact*: {
   -> formal_name* (String)
   -> email* (String)
   -> phone (String)
  }
  -> addresses: [
   -> street_line_1* (String)
   -> street_line_2 (String)
   -> city* (String)
   -> postal_code* (String)
   -> state_code* (String)
   -> country_code* (String)
   -> type* (String) #{shipping, billing}
  ]
  -> role* (String) #{seller}
 ]
]
```

Response

```
[
 <- uid (String)
 <- contact: {
  <- formal_name (String)
  <- email (String)
  <- phone (String)
 }
 <- addresses: [
  <- uid (String)
  <- street_line_1 (String)
  <- street_line_2 (String)
  <- city (String)
  <- postal_code (String)
  <- state_code (String)
```

```
   <- country_code (String)
   <- type (String) #{shipping, billing}
  ]
  <- role (String)
]
```

Payload for Updating Accounts: This API has the flexibility of processing data by a row or in bulk.

Update Accounts

Request

```
[
  -> marketplace_uid* (String)
  -> data*: [
    -> uid* (String)
    -> contact*: {
      -> formal_name* (String)
      -> email* (String)
      -> phone (String)
    }
    -> addresses: [
      -> street_line_1* (String)
      -> street_line_2 (String)
      -> city* (String)
      -> postal_code* (String)
      -> state_code* (String)
      -> country_code* (String)
      -> type* (String) #{shipping, billing}
    ]
    -> role* (String) #{seller}
  ]
]
```

Response

```
[
  <- uid (String)
  <- contact: {
    <- formal_name (String)
    <- email (String)
    <- phone (String)
  }
  <- addresses: [
    <- uid (String)
    <- street_line_1 (String)
    <- street_line_2 (String)
    <- city (String)
```

```
    <- postal_code (String)
    <- state_code (String)
    <- country_code (String)
    <- type (String) #{shipping, billing}
  ]
  <- role (String)
]
```

Product Availability Rules Endpoints (Exclusionary)

Get PAV Rules List

Request:

```
1{
2  -> supplier_uid* (String)
3  -> page (Int)
4  -> per_page (Int)
5}
```

Response:

```
1{
2  <- data: [
3    <- uid (String)
4    <- brand_uid (String)
5    <- zip_code (String)
6  ]
7  <- page (Int)
8  <- per_page (Int)
9  <- total_pages (Int)
10}
```

Create PAV Rules (Batch)

Request

```
1{
2  -> supplier_uid* (String)
3  -> data*: [
4    -> brand_uid* (String)
```

```
5    -> zip_code* (String)
6  ]
7}
```

Response

```
1[
2  <- uid (String)
3  <- brand_uid (String)
4  <- zip_code (String)
5]
```

Update PAV Rules (Batch)

Request

```
1{
2  -> supplier_uid* (String)
3  -> data*: [
4    -> brand_uid* (String)
5    -> zip_code* (String)
6  ]
7}
```

Response

```
1[
2  <- uid (String)
3  <- brand_uid (String)
4  <- zip_code (String)
5]
```

Order Returns Endpoint

Request

{

-> marketplace_uid* (String)

-> data*: [

-> order_uid* (String)

-> reason (String)

-> comment (String)

]

}

Response

[

<- order_uid (String)

<- shipping_status (String)

<- reason (String)

<- comment (String)

]

Additional Q&A

How do I update only product inventory?

Please use Products Update (Batch) endpoint with input limited to:

```
1{
2  -> supplier_uid* (String)
3  -> data*: [
4    -> uid* (String)
5    -> availability: {
6      -> availability_control (Bool)
7      -> available_quantity (Int)
8    }
9  ]
10}
```

How do I update only product descriptions?

Please use Products Update (Batch) endpoint with input limited to:

```
1{
2  -> supplier_uid* (String)
3  -> data*: [
4    -> uid* (String)
5    -> description (String)
6  ]
7}
```

How to update only shipping status for specified orders?

Please use Orders Update (Batch) endpoint, which allows you to change only shipping_status and tracking_url . To change only shipping status please limit input to:

1{

2 -> marketplace_uid* (String)

3 -> data: [

4 -> order_uid* (String)

5 -> shipping_status (String)

6]

7}