**Brand Qlik Cloud**

This allows you to:

* Replace the tenant Qlik logo with your own logo.
* Replace the tenant favicon with your own favicon.

Here is the Qlik dev site to help you learn more about Qlik Branding: [Link](https://qlik.dev/manage/platform-operations/brand-a-tenant/)

Here is the Qlik dev site to see what APIs we use for Qlik branding: [Link](https://qlik.dev/apis/rest/brands/)

**Introduction**

This document provides step-by-step instructions for branding a Qlik Cloud tenant using two approaches:

1. **Postman** - A manual API testing tool.
2. **Node.js Automation** - A programmatic approach for automation.

A close up of a logo

AI-generated content may be incorrect.

**Approach 1: Branding Using Postman**

Before starting, ensure you have:  
✔ **Admin access** to your Qlik Cloud tenant  
✔ **An API key** (generated in Qlik Cloud): [**Link**](https://qlik.dev/authenticate/api-key/generate-your-first-api-key/)  
✔ **Postman installed** on your machine:  [**Link**](https://www.postman.com/downloads/)

**Step 1️: Generate API Key in Qlik Cloud**

1. Log in to **Qlik Cloud**
2. Click on your **profile picture (top-right corner)** → **Profile Settings**
3. Go to **API Keys** tab
4. Click **"Generate New Key"**
   * Set an expiration date
   * Give it a name (e.g., Branding API Key)
5. **Copy the API key** and save it securely (you won't see it again)

**Step 2️: Set Up Authorization in Postman**

1. Open **Postman**
2. Click **"New Request"**
3. In the Authorization tab:
   * Select Bearer Token
   * Paste your API Key from Step 1 into the token field

**Step 3️: Upload Branding Assets (Logo, Favicon)**

1. **Set request type** to **POST**
2. **Enter the URL** (replace YOUR\_TENANT with your tenant domain)

https://YOUR\_TENANT/api/v1/brands

1. **Go to Headers** tab:
   * **Key:** Authorization
   * **Value:** Bearer YOUR\_API\_KEY
2. **Go to Body** tab:
   * Select **form-data**
   * Add the following fields:
     + name → **Your brand name**
     + description → **Brief brand description**
     + logo → **Upload your logo file (PNG or JPG)**
     + favIcon → **Upload your favicon (ICO or PNG)**
3. Click **Send**

**Response Example (Success)**

{

"id": "67a979d232d11626d70a416a",

“active”: false

"name": "My Custom Brand",

"description": "Branding for My Company",

}

**Copy the id from the response – you’ll need it in the next step.**

**Step 4️: Activate the Brand**

1. **Set request type** to **POST**
2. **Enter the URL** (replace BRAND\_ID with the ID from Step 3)

https://YOUR\_TENANT/api/v1/brands/BRAND\_ID/actions/activate

1. **Go to Headers** tab:
   * **Key:** Authorization
   * **Value:** Bearer YOUR\_API\_KEY
2. Click **Send**

**Response Example (Success)**

{

"id": "67a979d232d11626d70a416a",

“active”: true

"name": "My Custom Brand",

}

**Your brand is now active!**

**Step 5️: Verify Branding in Qlik Cloud**

1. Open **Qlik Cloud**
2. Refresh the page
3. Your **logo, favicon** should now be applied

**Summary**

| **Step** | **Action** |
| --- | --- |
| **1️. Generate API Key** | Get an API key from Qlik Cloud |
| **2️. Set Up Postman** | Use **Bearer Token** authentication |
| **3️. Upload Branding Assets** | Send a POST request with logo, favicon, and styles |
| **4️. Activate the Brand** | Send a POST request to activate branding |
| **5️. Verify Branding** | Refresh Qlik Cloud and check your custom branding |

**Next Steps**

* **Modify Branding**: Re-run Step 3 with different assets
* **Deactivate a Brand**: Send a POST request to

https://YOUR\_TENANT/api/v1/brands/BRAND\_ID/actions/deactivate

* **List All Brands**:

GET <https://YOUR_TENANT/api/v1/brands>

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Using **Postman** for Qlik **tenant branding** has several disadvantages compared to a **Node.js automation script**. Here are the key drawbacks:

**1. Manual Process – No Automation**

* Each **API request** (creating a brand, uploading a logo, activating the brand, updating styles) **must be done manually**.
* If you have **multiple clients**, you’ll need to **repeat the process for each tenant**.

**2. Difficult to Scale for Multiple Clients**

* For every new client, you **must manually input the branding details** in Postman.
* In contrast, a **Node.js script can loop through a JSON file** and apply branding automatically.

**3. No Error Handling & Logging**

* If an API call **fails in Postman**, you must manually check logs.
* With Node.js, you can **log failures automatically and retry failed requests**.

**4. Harder to Manage API Keys Securely**

* In Postman, **API keys are stored as environment variables** but are still visible.
* With Node.js, API keys can be stored in a **.env file**, keeping them more secure.

**5. No Scheduled or Continuous Updates**

* If a client **wants to update their branding later**, you must manually update and resend API requests.
* A **script can check for updates regularly and apply changes automatically**.

**6. No Database/JSON Storage for Brand IDs**

* After creating a brand in Postman, you **must manually copy and store the brand ID**.
* In Node.js, the script **automatically updates the clients.json file**, storing brand IDs and statuses.

**7. No Conditional Logic**

* In Postman, you can’t easily check:
  + If a brand **already exists** before creating a new one.
  + If a brand **is activated before applying styles**.
* A **script can handle this logic efficiently**.

**Conclusion: Node.js is Better for Automation**

Using **Node.js** for Qlik **tenant branding** is more efficient, scalable, and secure than using Postman. Postman is good for **one-time manual testing**, but for **multiple clients and ongoing branding updates**, automation with Node.js is recommended approach.

**Approach 2: Automated Branding with Node.js**

**Step 1: Install Node.js and VSCode**

1. Download and install **Node.js** from <https://nodejs.org>.
2. Download and install **Visual Studio Code** from [https://code.visualstudio.com](https://code.visualstudio.com/).
3. Create a .env file for API keys.

**Folder Structure**

branding-automation/

│── clients.json

│── .env

│── index.js

│── assets/

│── logos/

│── favicons/

│── styles/

**Step 2: Set Up Your Project**

1. Open VSCode and create a new folder for your project.
2. Open a terminal in VSCode and run:

npm init -y

This will create a package.json file.

1. Install required dependencies:

npm install dotenv axios form-data fs

**Step 3: Configure API Keys and Environment Variables**

1. Create a .env file in your project directory and add:

QLIK\_API\_KEY=your\_api\_key\_here

Replace your\_api\_key\_here with the actual Qlik API key.

**Step 4: Create clients. json File**

1. In the project folder, create a clients.json file and structure it like this:

json

CopyEdit

{

"clients": [

{

"name": "Paratek",

"tenant\_url": "karthikburra93.us.qlikcloud.com",

"api\_key\_env": "QLIK\_API\_KEY",

"logo": "logos/ParatekLogo.png",

"favicon": "favicons/ParatekFavicon.ico",

"styles": "styles.json",

"brand\_id": "",

"activated": false

},

{

"name": "OccamHealth",

"tenant\_url": "occamhealth.us.qlikcloud.com",

"api\_key\_env": "QLIK\_API\_KEY",

"logo": "logos/OccamLogo.png",

"favicon": "favicons/OccamFavicon.ico",

"styles": "styles.json",

"brand\_id": "",

"activated": false

}

]

}

* + api\_key\_env: Specifies which key to fetch from .env.
  + brand\_id: Will be updated after brand creation.
  + activated: Will be set to true after activation.

**Step 5: Create branding.js**

1. In the project folder, create branding.js and add the following:

require("dotenv").config();

const https = require("https");

const fs = require("fs");

const FormData = require("form-data");

// Load clients from JSON

const clientsFile = "clients.json";

let clientsData = JSON.parse(fs.readFileSync(clientsFile, "utf-8"));

// Function to get API key from `.env` based on `api\_key\_env` field

function getApiKey(client) {

  if (!client.api\_key\_env) {

    console.error(`No API key reference found for ${client.name}.`);

    return null;

  }

  const apiKey = process.env[client.api\_key\_env];

  if (!apiKey) {

    console.error(

      `API key ${client.api\_key\_env} is missing in .env for ${client.name}.`

    );

  }

  return apiKey;

}

// Function to create a brand

function createBrand(client) {

  return new Promise((resolve, reject) => {

    const { name, tenant\_url, logo, favicon, styles, activated } = client;

    console.log(client);

    const apiKey = getApiKey(client);

    if (!fs.existsSync(logo) || !fs.existsSync(favicon)) {

      console.error(`Missing branding files for ${name}.`);

      return resolve(null);

    }

    // Create form-data request

    const formData = new FormData();

    formData.append("name", name);

    formData.append("description", `Branding for ${name}`);

    formData.append("logo", fs.createReadStream(logo));

    formData.append("favIcon", fs.createReadStream(favicon));

    formData.append("styles", fs.createReadStream(styles)); // Brand Styles

    const options = {

      hostname: tenant\_url,

      port: 443,

      path: "/api/v1/brands",

      method: "POST",

      headers: {

        Authorization: `Bearer ${apiKey}`,

        ...formData.getHeaders(),

      },

    };

    const req = https.request(options, (res) => {

      let data = "";

      res.on("data", (chunk) => {

        data += chunk;

      });

      res.on("end", () => {

        try {

          const responseData = JSON.parse(data);

          if (res.statusCode === 201) {

            console.log(

              `Brand Created for ${name}. Brand ID: ${responseData.id}`

            );

            resolve(responseData.id);

          } else {

            console.error(`Failed for ${name}:`, responseData);

            resolve(null);

          }

        } catch (error) {

          console.error(`Error parsing response for ${name}:`, error);

          resolve(null);

        }

      });

    });

    req.on("error", (error) => {

      console.error(`Request Error for ${name}:`, error.message);

      resolve(null);

    });

    formData.pipe(req);

  });

}

// Function to activate a brand

function activateBrand(client, brandId) {

  const apiKey = getApiKey(client);

  return new Promise((resolve, reject) => {

    const options = {

      hostname: client.tenant\_url,

      port: 443,

      path: `/api/v1/brands/${brandId}/actions/activate`,

      method: "POST",

      headers: {

        Authorization: `Bearer ${apiKey}`,

        "Content-Type": "application/json",

      },

    };

    const req = https.request(options, (res) => {

      let data = "";

      res.on("data", (chunk) => {

        data += chunk;

      });

      res.on("end", () => {

        if (res.statusCode === 200) {

          console.log(`Brand Activated for ${client.name}`);

          resolve(true);

        } else {

          console.error(`Activation Failed for ${client.name}:`, data);

          resolve(false);

        }

      });

    });

    req.on("error", (error) => {

      console.error(`Request Error for ${client.name}:`, error.message);

      resolve(false);

    });

    req.end();

  });

}

// Function to process all clients

async function processClients() {

  for (let client of clientsData.clients) {

    if (!client.brand\_id) {

      const brandId = await createBrand(client);

      console.log("Brand ID: ", brandId);

      if (brandId) {

        client.brand\_id = brandId;

        const isActivated = await activateBrand(client, brandId);

        client.activated = isActivated;

      }

    } else {

      console.log(

        `Brand already exists for ${client.name}. Brand ID: ${client.brand\_id}`

      );

    }

  }

  // Save updated client data with brand IDs

  fs.writeFileSync(clientsFile, JSON.stringify(clientsData, null, 4), "utf-8");

  console.log("All clients processed & saved.");

}

// Run the automation

processClients();

**Step 6: Run the Automation**

1. In the terminal, run:

node branding.js

1. This will:
   * Read the client list from clients.json.
   * Create a brand for each client (if not already created).
   * Activate the brand.
   * Update clients.json with brand\_id and activation status.

**Step 7: Maintain and Update Branding**

* To update branding (e.g., new logo), replace the logo files and **delete the brand\_id** from clients.json, then rerun the script.
* To add new clients, just add them to clients.json and run the script.

**Advantages of This Approach**

✔ **Scalable** – Supports multiple clients.  
✔ **Automated** – No need for manual API calls.  
✔ **Secure** – API keys are stored in .env, not in clients.json.  
✔ **Easy Maintenance** – Modify branding by updating files and rerunning.