# QA Portafolio Example

**Katy Silva**

# Part 1: Web Application Testing

## Identify and Write Test Cases

### 1. Test Case: Guide Completion Status Update

**Objective:** Verify that marking a guide as “done” updates the completion status correctly.
**Steps:** Login, navigate to the “Guides” section, select a guide, mark as “done”, and verify the count increments appropriately on the dashboard.
**Expected Result:** Guides completed count should increment by one.

### 2. Test Case: Guide Flagging Functionality

**Objective:** Ensure that flagging a guide does not increment the “guidesCompleted” count.
**Steps:** Login, select a guide, mark as “flagged”, and verify the count remains unchanged on the dashboard.
**Expected Result:** Guides completed count should not change.

### 3. Test Case: Responsive Design Check

**Objective:** Confirm that the web application's layout adjusts correctly to different screen sizes.
**Steps:** Open the web application on various devices and browsers, check layout and functionality.
**Expected Result:** UI elements should be well-aligned and functional across devices.

## 4. Test Case: Access Control Verification

**Objective:** Validate that users can only access device data associated with their account.
**Steps:** Login with different user credentials and attempt to access device data.
**Expected Result:** Users should only see data for devices linked to their account.

## 5. Test Case: Header Filter Functionality

**Objective:** Ensure the header filters correctly sort the device list based on the selected criteria.
**Steps:** Login, apply each header filter (*All devices:* Enrolled, Not Enrolled; *Severity levels:* Critical, Fixes needed, Secure), and observe the changes in the device list.
**Expected Result:** The device list should update to reflect the filter criteria accurately, displaying the appropriate subset of devices for each filter.

## 6. Test Case: Dashboard Load Performance

**Objective:** Assess the loading time of the dashboard with multiple devices listed.
**Steps:** Login, navigate to the dashboard with a known number of devices, measure load time.
**Expected Result:** The dashboard should load within an acceptable time frame, adhering to performance benchmarks.

Bug Report

## BE - Dashboard Metric Discrepancy: “Flagged” Guides Incorrectly Counted as “Completed”

**Device / Browser & operating system:**
MacOS Sonoma Version 14.2.1 (23C71)
Chrome Version 120.0.6099.234 (Official Build) (arm64)
&
iPhone 11 Pro Max (iOS 16.1.1)
**Affects versions:** 41.0 PROD
**Username & password used:** testing123@gmail.com pass123

The current BE implementation increments the “guidesCompleted” counter for both “done” and “flagged” statuses. This conflates two distinct user actions, potentially skewing the actual completion metrics. For clarity and accuracy in reporting, “flagged” items should not be considered as “completed” guides.

**Requirements:**
Users must log in to the platform using an account linked to a device that haven't completed all guides.

**Steps to reproduce:**

1. Log in to https://portal.iverify.io/dashboard .
2. Open iVerify for Organizations mobile app.
3. Tap on “Guides”
4. Select one that has not been completed yet.
5. Scroll to the end of the guide and tap on the “Flag for later” option.
6. Go to the dashboard.
7. Refresh.

Notice that “Flag for later” action is increasing the number of completed guides on the dashboard.

Notes:

1. **Guide Status Handling:** When a “Flagged” guide is marked as “Done”, do not overcount, ensuring the count does not exceed the available number of guides (36).

2. **Irreversibility of “Done” Status:** Currently, there is no feature allowing users to mark a “Done” guide as incomplete again. This limitation restricts users from correcting their progress if they've mistakenly marked a guide as completed.
   Also, after the guide is “done”, when passing to “flagged”, the count does not decrease.

3. **API and Front-End Discrepancy:** The API considers both “Done” and “Flagged” statuses as completed, which contrasts with the front-end representation on iOS devices. The front-end representation on iOS devices seems to be interpreting and displaying this data differently, perhaps filtering out “flagged” from “completed” visually. It may not be an immediate user-facing issue, but it's a technical debt that can lead to confusion or errors in the future.

**Expected vs actual results:**

**Expected:** Flagging a guide for later should not increment the “guidesCompleted” count within the dashboard metrics.

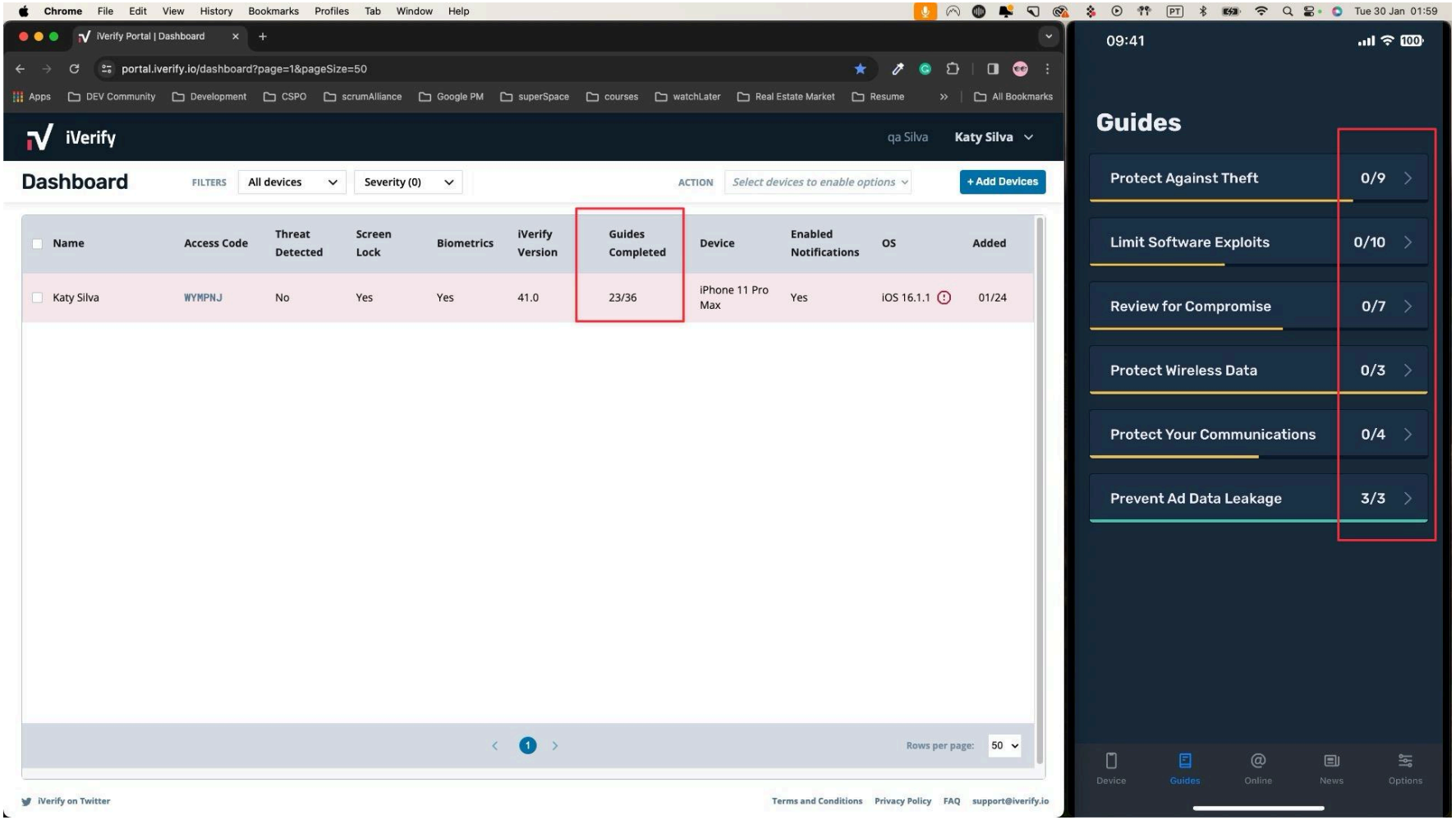
**Actual:** The dashboard increments the “guidesCompleted” count when a guide is flagged, conflating this status with completion.

**Recommendations:**

- Conduct a comprehensive audit of both front-end and back-end logic related to guide status tracking.
- Consider implementing a mechanism to allow users to mark a guide as “incomplete” after setting it to “done.”

**Severity level:**

**Medium-High:** While it does not impact the immediate functionality, it poses significant risks to data accuracy and user trust in the long term.


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Tue 30 Jan 01:59
iVerify Portal | Dashboard
portal.iverify.io/dashboard?page=1&pageSize=50
Apps DEV Community Development CSPO scrumAlliance Google PM superSpace courses watchLater Real Estate Market Resume All Bookmarks
iVerify
qa Silva
Katy Silva
Dashboard
FILTERS
All devices
Severity (0)
ACTION
Select devices to enable options
+ Add Devices
Name
Access Code
Threat Detected
Screen Lock
Biometrics
iVerify Version
Guides Completed
Device
Enabled Notifications
OS
Added
Katy Silva
WYMPNJ
No
Yes
Yes
41.0
23/36
iPhone 11 Pro Max
Yes
iOS 16.1.1
01/24
Rows per page:
50
iVerify on Twitter
Terms and Conditions
Privacy Policy
FAQ
support@iverify.io
09:41
Guides
Protect Against Theft 0/9
Limit Software Exploits 0/10
Review for Compromise 0/7
Protect Wireless Data 0/3
Protect Your Communications 0/4
Prevent Ad Data Leakage 3/3
Device
Guides
Online
News
Options

# Part 2: Mobile Application Testing

## Test Plan Outline for iVerify for Organizations App:

### 1. Functional Testing:

**Login/Auth:** Test the login flow using the provided access code.
**Guide Interaction:** Verify the functionality of completing, flagging, and accessing guides.
**Dashboard:** Ensure the dashboard accurately reflects changes made within the app.

### 2. Usability Testing:

**Navigation:** Confirm intuitive navigation through the app's features.
**Accessibility:** Check for font sizes, color contrasts, and voiceover compatibility.
**Error Handling:** Test app behavior on incorrect input or actions.

### 3. Compatibility Testing:

**Multiple Devices:** Test on various devices with different screen sizes and OS versions.
**Orientation:** Check app behavior in both landscape and portrait modes.

4. Performance Testing:

**Load Time:** Measure app responsiveness and load times for different functions.
**Resource Usage:** Monitor the app's usage of system resources, such as CPU and memory.

5. Security Testing:

**Data Protection:** Verify that user data is securely handled and stored.
**Penetration Testing:** Test for vulnerabilities to unauthorized access *(Ex. SQL injection)*.

6. Network Testing:

**Offline Functionality:** Determine app behavior with no internet access.
**Network Conditions:** Test app performance across different network speeds.

## Tools and Approach:

For automated testing of the mobile application, tools like Appium could be used.
Appium will be utilized to automate functional testing of the iVerify mobile app, including interactions with the app's API. The tool allows for scripting sequences of API calls that simulate user actions, such as logging in, retrieving data, and updating settings. This can help ensure the mobile app interacts correctly with the backend services, maintaining consistency across different devices and operating systems. Appium's ability to support both Android and iOS platforms makes it an ideal choice for cross-platform mobile application testing.

The approach would include:

- Writing test scripts that mimic user actions and verify the expected outcomes.
- Integrating with a Continuous Integration (CI) system to execute tests on various devices.
- Utilizing cloud-based device farms like BrowserStack or Sauce Labs for testing on a wide range of devices and configurations.
- For manual testing aspects, such as usability, tester would evaluate the app's interface and user flows.

Additionally, understanding the expected behavior and business logic from the product documentation would be essential to select the correct approach.

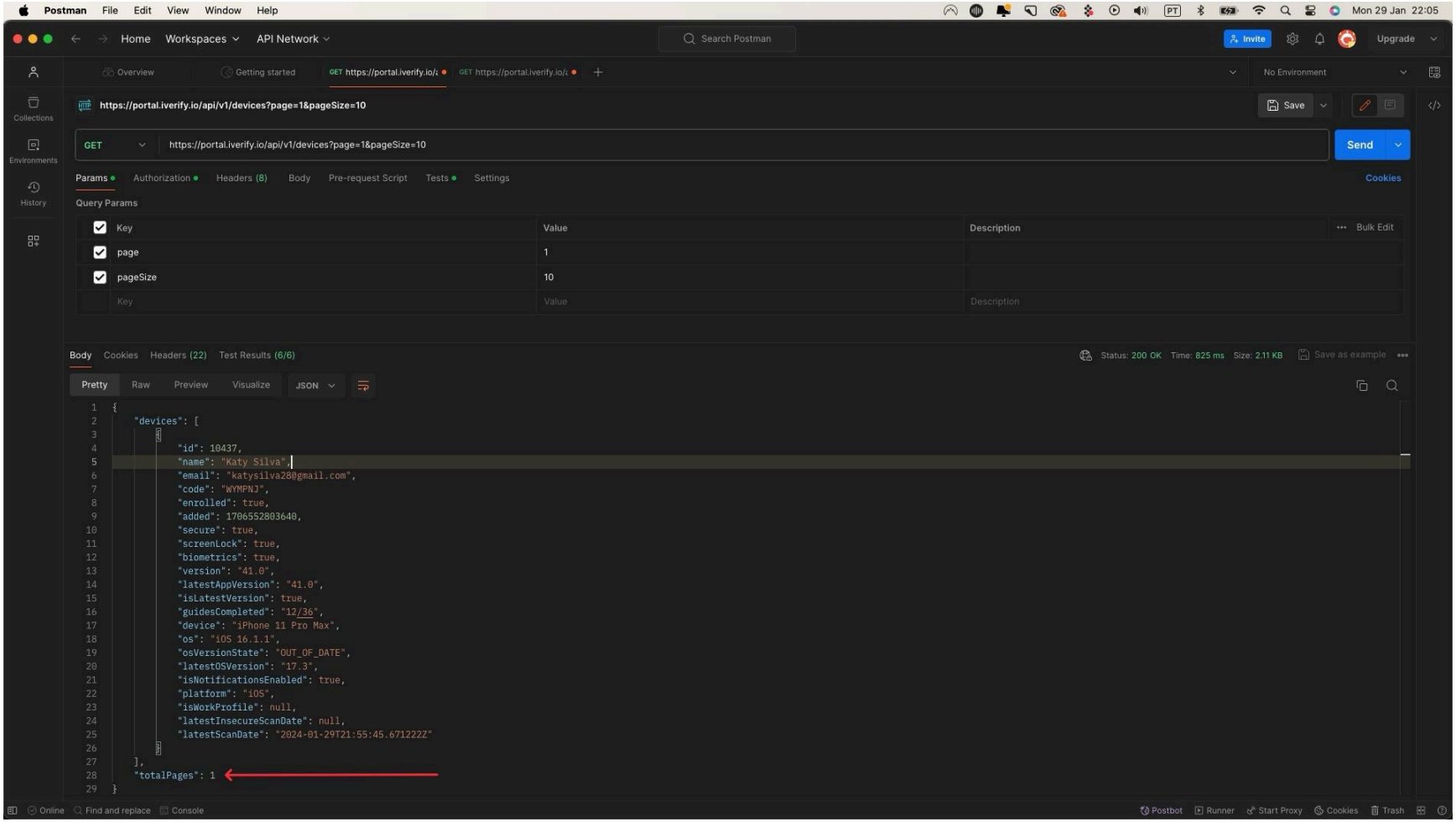
# Part 3: API Testing

For API testing, I utilized Postman, and I wrote a series of JavaScript tests using the pm object to programmatically ensure that the API's responses met our expectations. This combination of tools allows for robust testing of API endpoints covering various HTTP methods, response validations, and error handling.

## GET Request for Device List (Pagination)

- **Objective**: Verify the API correctly handles pagination in the device list.
- **Method**: GET
- **Endpoint**: https://portal.iverify.io/api/v1/devices?page=1&pageSize=10
- **Expected Status Code**: 200 OK
- **Expected Response**: JSON object containing a list of up to 10 devices.
    - **Test Steps**:
    - Send a GET request with pagination parameters.
    - Validate the response code and JSON structure.
    - Ensure the number of devices returned does not exceed pageSize.

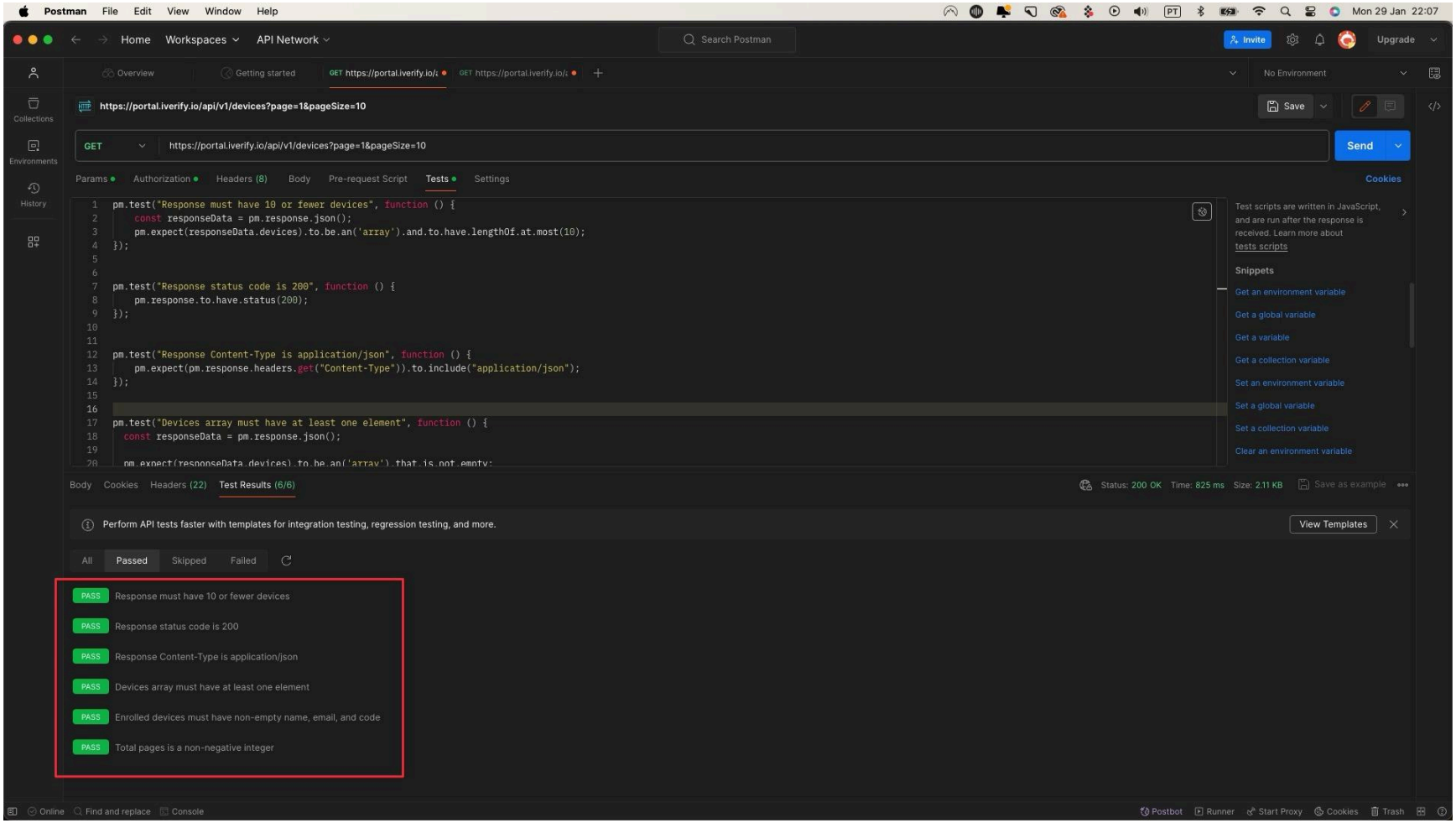
Postman:


Postman File Edit View Window Help
Mon 29 Jan 22:05
Home Workspaces API Network
Search Postman
Invite
Upgrade
Overview
Getting started
GET https://portal.iverify.io/a
GET https://portal.iverify.io/a
No Environment
Collections
Environments
History
https://portal.iverify.io/api/v1/devices?page=1&pageSize=10
Save
GET
https://portal.iverify.io/api/v1/devices?page=1&pageSize=10
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
page
1
pageSize
10
Key
Value
Description
Body
Cookies
Headers (22)
Test Results (6/6)
Status: 200 OK
Time: 825 ms
Size: 2.11 KB
Save as example
Pretty
Raw
Preview
Visualize
JSON
{
"devices": [
{
"id": 10437,
"name": "Katy Silva",
"email": "katysilva28@gmail.com",
"code": "WYMPNJ",
"enrolled": true,
"added": 1706552803640,
"secure": true,
"screenLock": true,
"biometrics": true,
"version": "41.0",
"latestAppVersion": "41.0",
"isLatestVersion": true,
"guidesCompleted": "12/36",
"device": "iPhone 11 Pro Max",
"os": "iOS 16.1.1",
"osVersionState": "OUT_OF_DATE",
"latestOSVersion": "17.3",
"isNotificationsEnabled": true,
"platform": "iOS",
"isWorkProfile": null,
"latestInsecureScanDate": null,
"latestScanDate": "2024-01-29T21:55:45.671222Z"
}
],
"totalPages": 1
}
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Trash

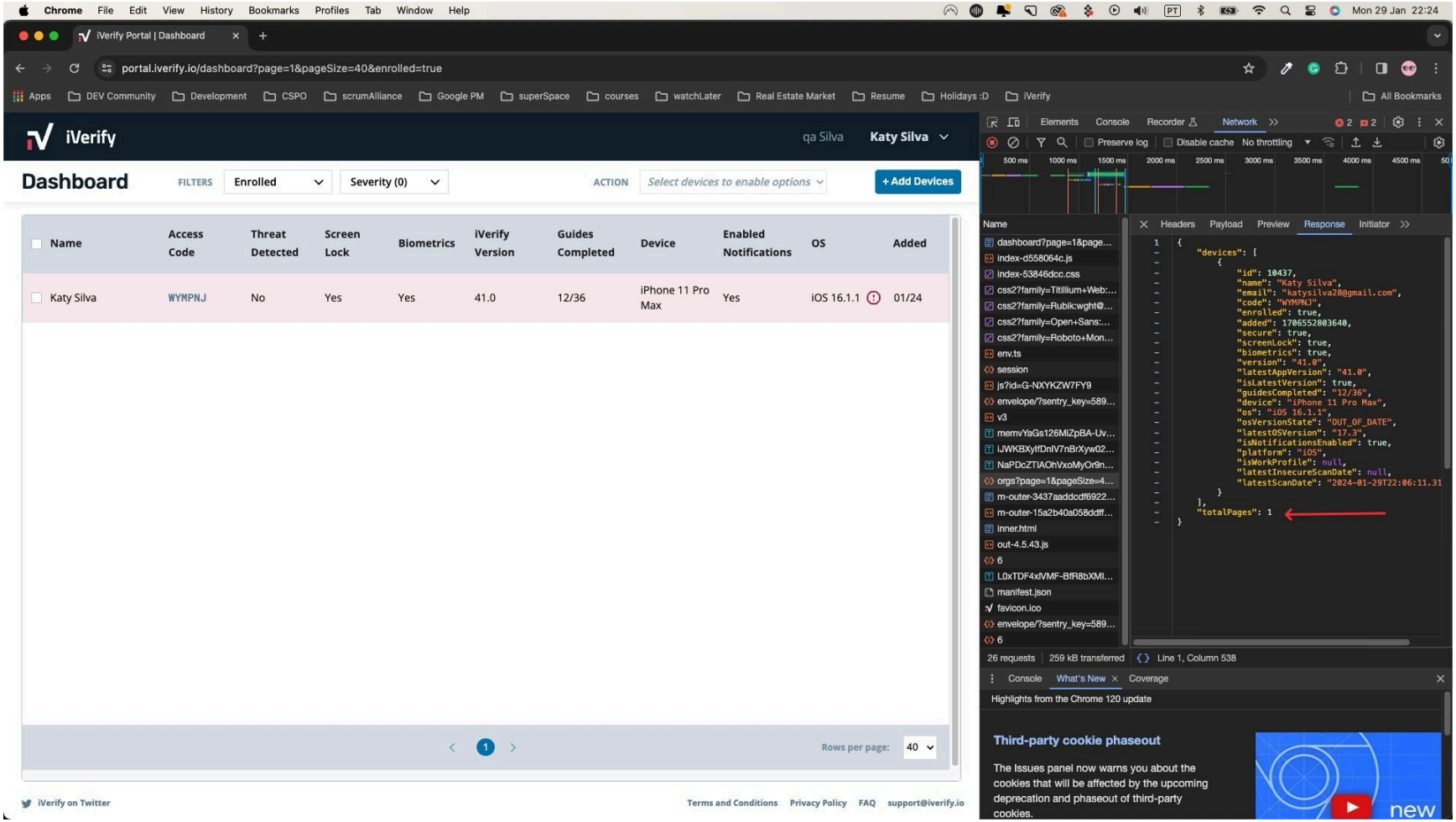
Automated Testing Script:

```
pm.test("Response must have 10 or fewer devices", function () {
   const responseData = pm.response.json();
   pm.expect(responseData.devices).to.be.an('array').and.to.have.lengthOf.at.most(10);
});


pm.test("Response status code is 200", function () {
   pm.response.to.have.status(200);
});


pm.test("Response Content-Type is application/json", function () {
   pm.expect(pm.response.headers.get("Content-Type")).to.include("application/json");
});


pm.test("Devices array must have at least one element", function () {
 const responseData = pm.response.json();
  pm.expect(responseData.devices).to.be.an('array').that.is.not.empty;
});


pm.test("Enrolled devices must have non-empty name, email, and code", function () {
   const responseData = pm.response.json();

   pm.expect(responseData.devices).to.be.an('array').that.is.not.empty;
```

```
    responseData.devices.forEach(function(device) {
        pm.expect(device.name).to.be.a('string').that.is.not.empty;
        pm.expect(device.email).to.be.a('string').that.is.not.empty;
        pm.expect(device.code).to.be.a('string').that.is.not.empty;
    });
});


pm.test("Total pages is a non-negative integer", function () {
 const responseData = pm.response.json();
  pm.expect(responseData.totalPages).to.be.a('number');
 pm.expect(responseData.totalPages).to.be.at.least(0, "Total pages should be non-negative");
});
```

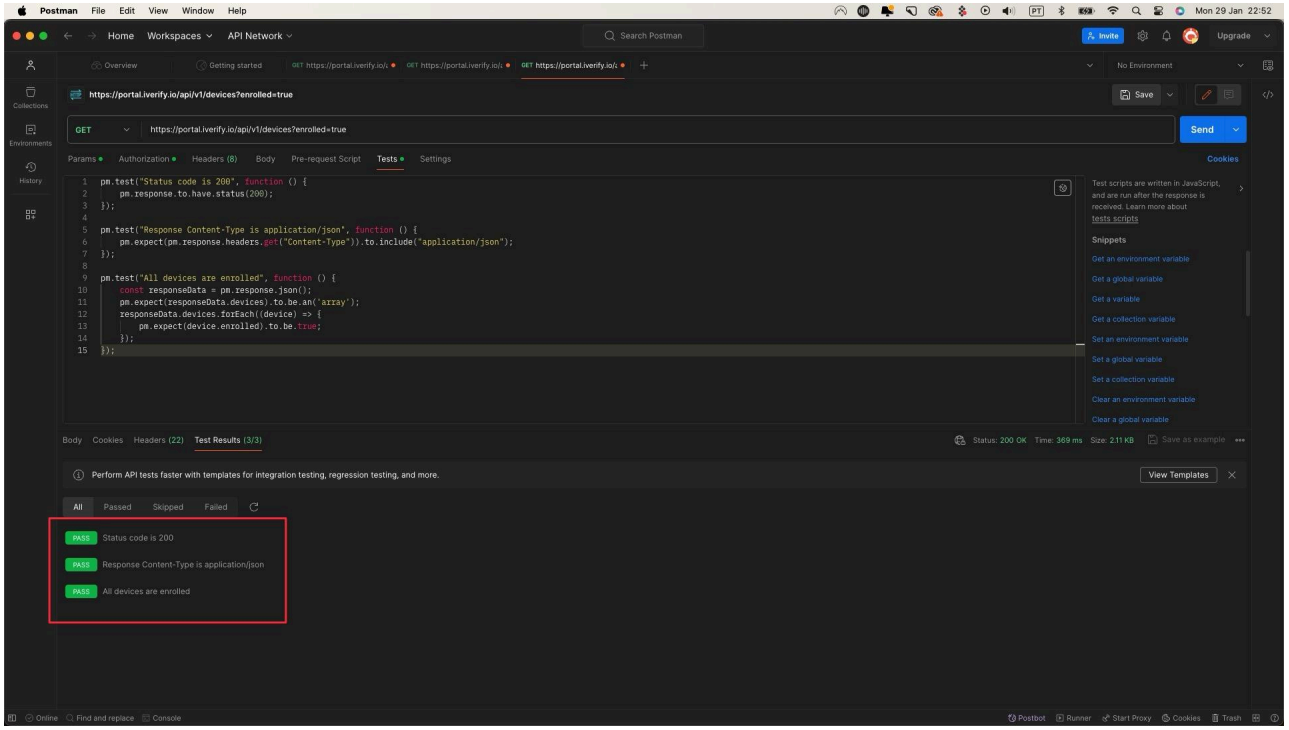

Postman File Edit View Window Help
Mon 29 Jan 22:07
Home Workspaces API Network
Search Postman
Invite
Upgrade
Overview
Getting started
GET https://portal.iverify.io/a
GET https://portal.iverify.io/a
No Environment
Collections
Environments
History
https://portal.iverify.io/api/v1/devices?page=1&pageSize=10
Save
GET
https://portal.iverify.io/api/v1/devices?page=1&pageSize=10
Send
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
pm.test("Response must have 10 or fewer devices", function () {
const responseData = pm.response.json();
pm.expect(responseData.devices).to.be.an('array').and.to.have.lengthOf.at.most(10);
});
pm.test("Response status code is 200", function () {
pm.response.to.have.status(200);
});
pm.test("Response Content-Type is application/json", function () {
pm.expect(pm.response.headers.get("Content-Type")).to.include("application/json");
});
pm.test("Devices array must have at least one element", function () {
const responseData = pm.response.json();
Test scripts are written in JavaScript, and are run after the response is received. Learn more about tests scripts
Snippets
Get an environment variable
Get a global variable
Get a variable
Get a collection variable
Set an environment variable
Set a global variable
Set a collection variable
Clear an environment variable
Body
Cookies
Headers (22)
Test Results (6/6)
Status: 200 OK
Time: 825 ms
Size: 2.11 KB
Save as example
Perform API tests faster with templates for integration testing, regression testing, and more.
View Templates
All
Passed
Skipped
Failed
PASS Response must have 10 or fewer devices
PASS Response status code is 200
PASS Response Content-Type is application/json
PASS Devices array must have at least one element
PASS Enrolled devices must have non-empty name, email, and code
PASS Total pages is a non-negative integer
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Trash

Actual results:



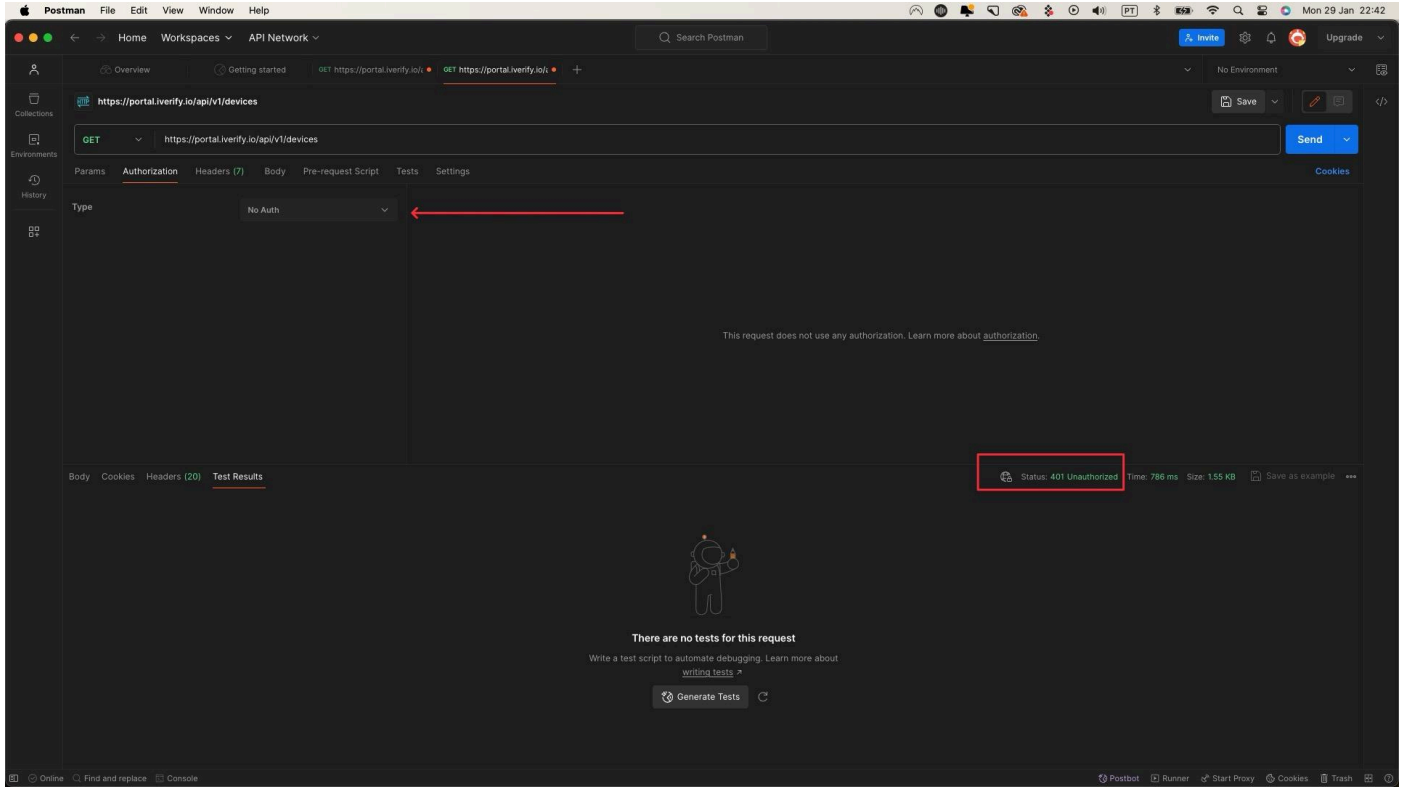
# GET Request for Filtered Device List

- **Objective**: Verify the API's ability to filter devices based on the 'enrolled' status.
- **Method**: GET
- **Endpoint**: https://portal.iverify.io/api/v1/devices?enrolled=true
- **Expected Status Code**: 200 OK
- **Expected Response**: JSON object with devices that are enrolled.
    - **Test Steps**:
    - Send a GET request with the enrolled filter.
    - Validate response status and JSON content.
    - Check that all returned devices have 'enrolled' status as true.



# Unauthorized Access Test

- **Objective**: Ensure the API denies access without a valid token.
- **Method**: GET
- **Endpoint**: https://portal.iverify.io/api/v1/devices
- **Expected Status Code**: 401 Unauthorized
    - **Test Steps**:
    - Send a GET request without an authentication token.
    - Validate that the response code is 401.
    - Confirm the response message indicates an authorization error.



# POST Request Test Case: Creating a New Resource

**Objective:** Verify that the API allows the creation of a new resource, such as a new device.
**Method:** POST
**Endpoint:** https://portal.iverify.io/api/v1/devices
**Required Headers:**
Content-Type: application/json, Authorization: Bearer <token>
**Request Body:** A JSON object containing the necessary information for a new device.
**Expected Status Code:** 201 Created or 200 OK (depending on API specification)
**Expected Response:** JSON object detailing the created device.
**Test Steps:**

- Send a POST request to the endpoint with the required headers and JSON body.
- Validate that the response status code is 201 Created.
- Check the response body to ensure it contains the details of the created device.

This test case ensures that the API correctly handles the creation of new devices and responds with the appropriate status code and details of the newly created device.

```
pm.test("Successful creation status code", function () {
    pm.response.to.have.status(200);
});
pm.test("Response contains device details", function () {
    var responseData = pm.response.json();
    pm.expect(responseData).to.have.property("name");
});
```