**Lab Report — Search Commands Deep Dive: Part 2**

**Name:** Dhruvish
**Date:** October 25, 2025
**Platform:** Splunk Cloud Trial

---

**Objective**

This lab focused on expanding your Splunk Search Processing Language (SPL) skills using **eval-based functions** and **string manipulation helpers**.
You practiced using if(), case(), coalesce(), lower()/upper(), and len() to:

- Label and classify events
- Normalize inconsistent field values
- Create cleaner, more informative summaries

These functions are essential for refining data, preparing reports, and ensuring consistent field logic in dashboards and alerts.

---

**Tools Used**

- **Splunk Cloud Trial** (or Splunk Free)
- **Web Browser**

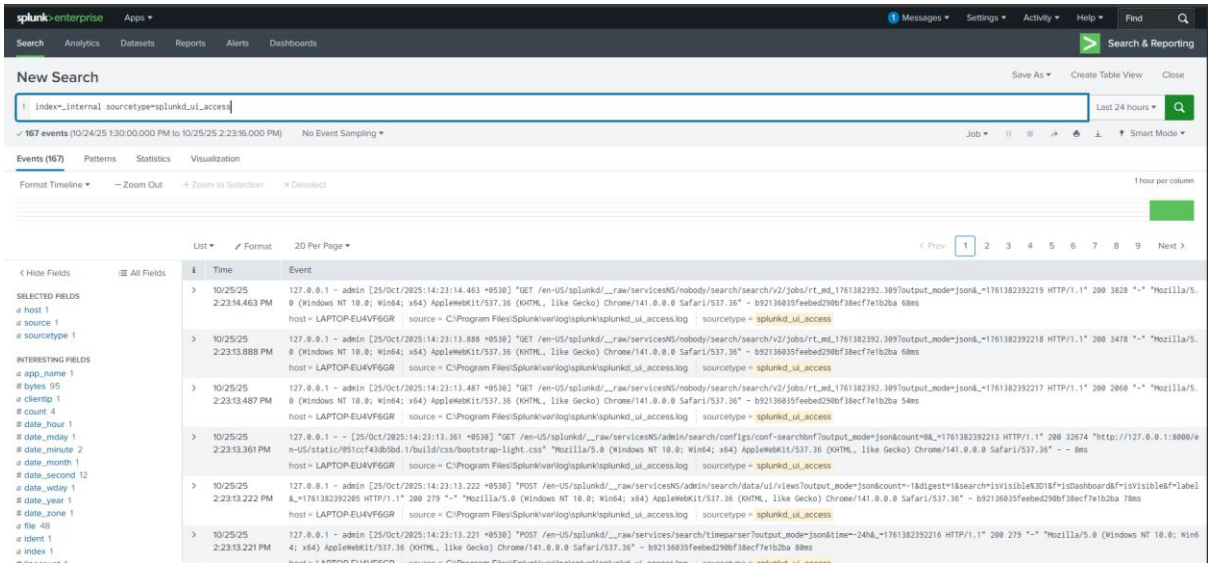
---

**Procedure and Observations**

**Step 1: Dataset & Time Setup**

- Opened **Apps ▸ Search & Reporting**.
- Set time range to **Last 24 hours**.
- Used the built-in dataset:

  index=_internal sourcetype=splunkd_ui_access
- Confirmed the presence of key fields: status, user, method, uri_path.



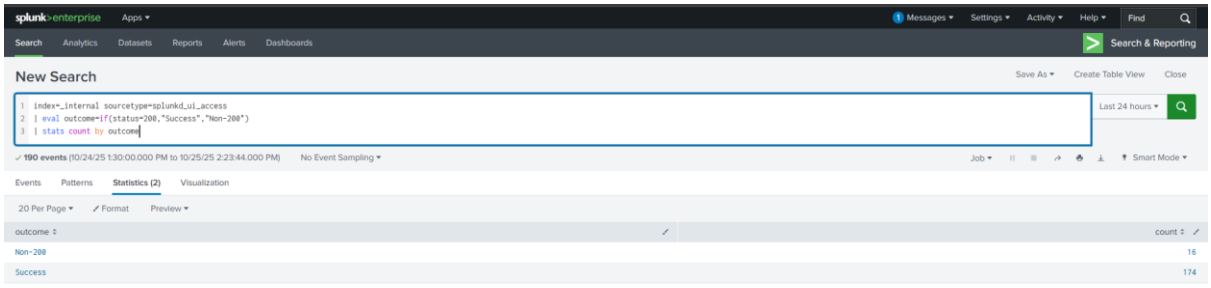

**Step 2: Label Events with if()**

Used conditional logic to label each event as a “Success” or “Non-200” based on HTTP status.

```
index=_internal sourcetype=splunkd_ui_access
| eval outcome=if(status=200,"Success","Non-200")
| stats count by outcome
```

Then filtered only failed outcomes:

```
... | where outcome="Non-200"
```




**Step 3: Multi-Branch Logic with case()**

Created HTTP class categories using case() for multi-condition evaluation:

```
index=_internal sourcetype=splunkd_ui_access
| eval http_class=case(
  status<200,"Informational",
  status>=200 AND status<300,"Success",
  status=404,"Not Found",
```

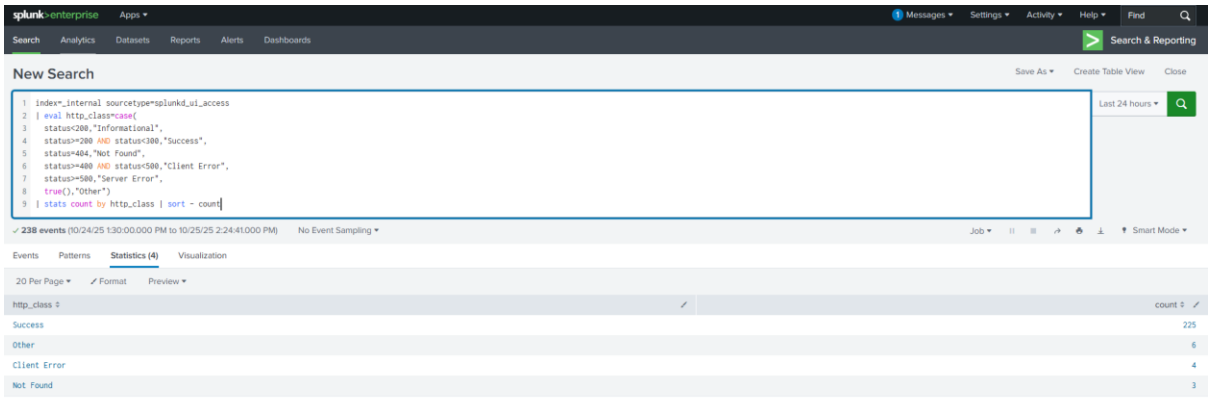
status>=400 AND status<500,"Client Error",

status>=500,"Server Error",

true(),"Other")

| stats count by http_class | sort - count

This allowed classification of status codes into meaningful HTTP response classes.




---

**Step 4: Normalize Identities with coalesce()**

Unified potentially inconsistent user fields using coalesce() to select the first available non-null value.

index=_internal sourcetype=splunkd_ui_access

| eval user_norm=coalesce(user, user_name, "-unknown-")

| stats count by user_norm | sort - count

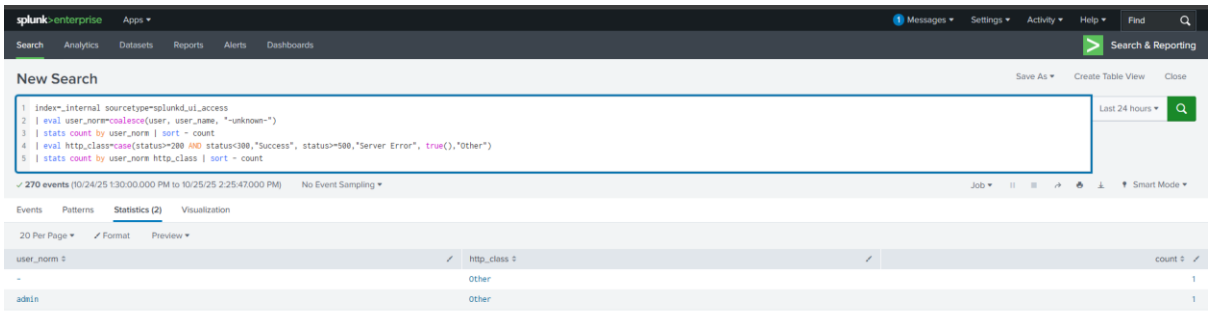
Then joined with class labeling:

... | eval http_class=case(status>=200 AND status<300,"Success",

status>=500,"Server Error",

true(),"Other")

| stats count by user_norm http_class | sort - count




---

**Step 5: Clean Keys with lower()/upper()**

Standardized usernames and methods to consistent casing for aggregation.

- **Lowercasing usernames:**

  index=_internal sourcetype=splunkd_ui_access

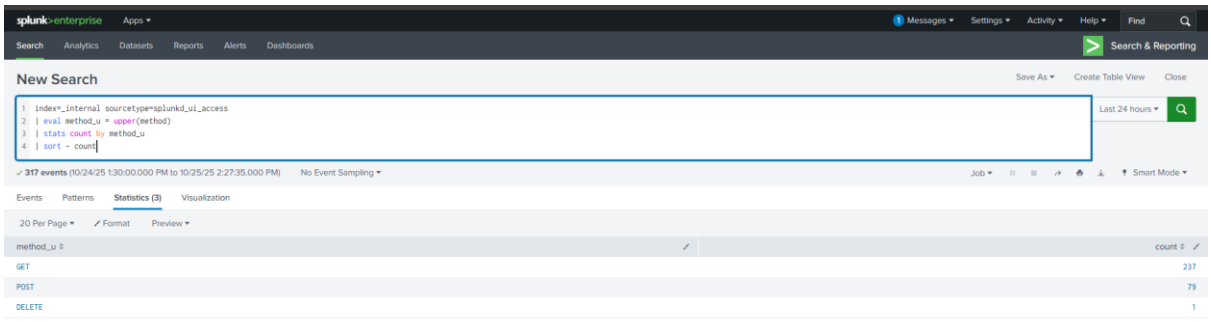
  | eval user_norm=lower(coalesce(user,user_name))

  | stats dc(uri_path) as unique_paths by user_norm | sort - unique_paths

- **Uppercasing HTTP methods:**

  ... | eval method_u=upper(method)

  | stats count by method_u




---

**Step 6: Spot Long URLs with len()**

Used len() to measure the length of URI paths and identify outliers.

index=_internal sourcetype=splunkd_ui_access

| eval url_len=len(uri_path)

| stats max(url_len) as max_len, avg(url_len) as avg_len by user

| sort - max_len

Filtered for URLs longer than 60 characters:

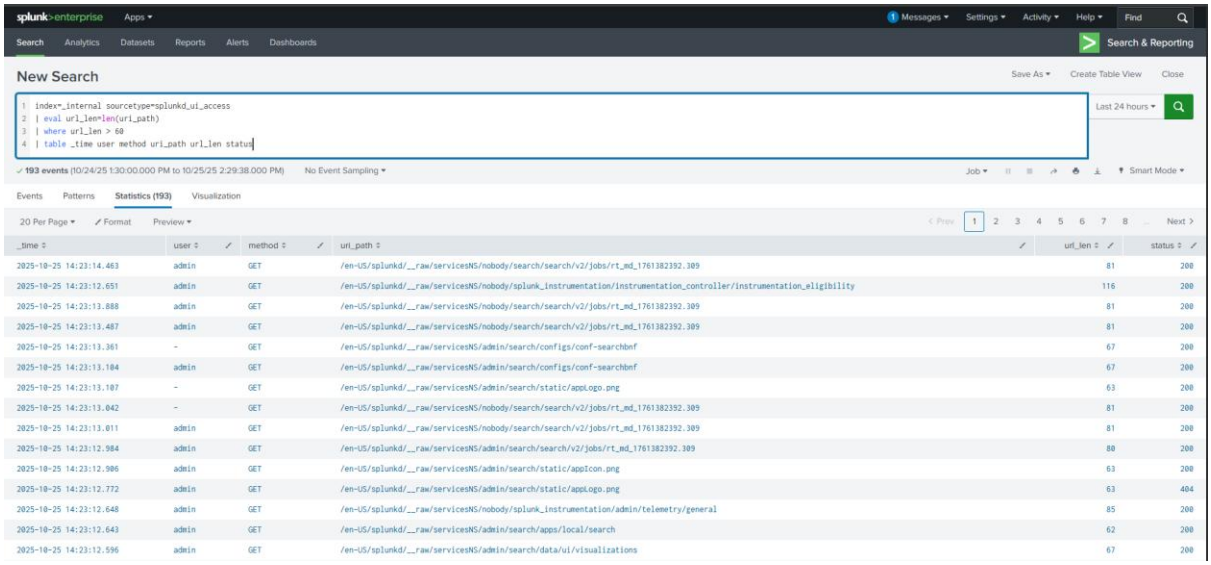
index=_internal sourcetype=splunkd_ui_access

| eval url_len=len(uri_path)

| where url_len > 60

| table _time user method uri_path url_len status

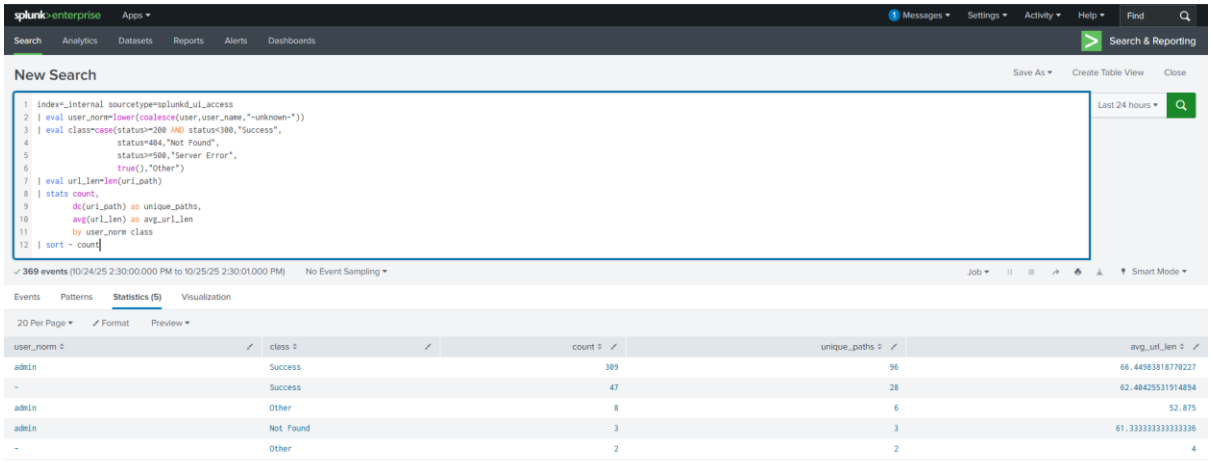
✅ *[Insert SS #6: Table showing long URL results (where url_len > 60)]*




**Step 7: Combined Summary Search**

Built a single unified search that combines labeling, normalization, and metrics:

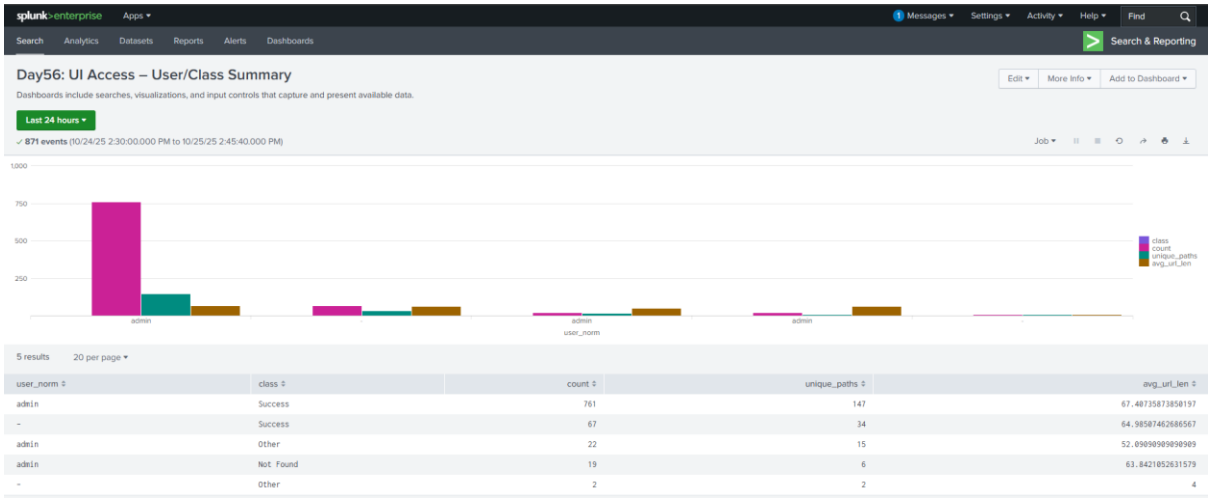
```
index=_internal sourcetype=splunkd_ui_access
| eval user_norm=lower(coalesce(user,user_name,"-unknown-"))
| eval class=case(status>=200 AND status<300,"Success",
                  status=404,"Not Found",
                  status>=500,"Server Error",
                  true(),"Other")
| eval url_len=len(uri_path)
| stats count,
    dc(uri_path) as unique_paths,
    avg(url_len) as avg_url_len
    by user_norm class
| sort - count
```




### Step 8: Save as a Report

Saved the final combined query as a reusable report for future dashboarding or alerting.

- **Report name:**
  *Day56: UI Access – User/Class Summary*
- **Location:** *Current App (Search & Reporting)*




### Reflection

- **Where did coalesce() help?**
  It unified user identity fields across different event types, avoiding missing or duplicated user counts when some logs used user and others user_name.
- **if() vs case():**
  Use if() for simple two-condition logic; use case() when classifying multiple conditions (e.g., HTTP status categories).
- **Impact of lower()/upper():**
  Normalizing case ensured accurate grouping. Without it, entries like *Admin*, *ADMIN*, and *admin* would be treated as separate users, skewing counts.

---

**Summary**

In this lab, I used **eval-based functions** to enhance Splunk search output by:

- Adding context and labels with if() and case()
- Cleaning inconsistent fields using coalesce()
- Standardizing text with lower() and upper()
- Measuring field lengths with len()
- Producing compact, accurate summaries ready for reports and dashboards

These SPL helpers form the backbone of efficient event normalization, data enrichment, and presentation within Splunk.