For the Surge Alerts microservice, consider and implement the below

NOTE: Check if there is an endpoint for hidden tweet stats already

Develop an API endpoint to fetch current throughput metrics related to hidden tweets and return the data to the frontend.

Endpoint Specification

Route:

GET /surge-alert/throughput/:brandId

Response Example:

{

"brand\_id": "123e4567-e89b-12d3-a456-426614174000",

"hidden\_replies\_last\_15\_min": 42,

"hidden\_replies\_last\_hour": 180,

"hidden\_replies\_last\_24\_hours": 1200

}

Copy

Wrap

Database Query:

SELECT

COUNT(\*) AS hidden\_replies\_last\_15\_min

FROM hidden\_tweets h

JOIN widgets w ON h.widget\_id = w.id

WHERE w.owner\_id = ?

AND h.hidden\_at >= NOW() - INTERVAL 15 MINUTE;

Copy

Wrap

Acceptance Criteria

Create an API endpoint to fetch hidden reply statistics.

Query and return hidden tweet throughput for the last 15 mins, hour, and 24 hours.

Ensure query performance by using indexes on hidden\_at.

More text incoming on

Create a **CRON job** that runs **every 1 minute** to check for **pending surge alerts** and trigger email notifications via the reply-manager-backend.

## **CRON Job Endpoint Logic**

1. **Fetch all unprocessed alerts (alerted\_at IS NULL)** by joining with reply\_manager\_surge\_alert\_config to get the latest config.
2. **If one or more alert entries exist**, compile the email subject and body, and send to all the emails configured
   1. Always BCC: reply-manager@bluerobot.com
3. **If the request is successful**, update alerted\_at in reply\_manager\_surge\_alerts to prevent duplicate alerts.

SIDE NOTE: We should be able to add all the configured emails to the BCC field to send the email to all configured addresses.

Database Query to Fetch Pending Alerts



SELECT rmsa.id, rmsa.surge\_alert\_config\_id, rmsa.surge\_amount,

rmsc.brand\_id, rmsc.emails\_to\_notify

FROM reply\_manager\_surge\_alerts rmsa

JOIN reply\_manager\_surge\_alert\_config rmsc

ON rmsa.surge\_alert\_config\_id = rmsc.id

WHERE rmsa.alerted\_at IS NULL;

## **Call the Reply Manager Backend API**

### **Endpoint:**

****

POST /surge-alert/notify

#### **Response Handling: (Note: we aren't sure what response structures we'll get)**

* ✅ **Success (200 OK)** → **Update alerted\_at**
* ❌ **Failure** → Log and retry in the next CRON execution.

### **Update alerted\_at on Success**

****

UPDATE reply\_manager\_surge\_alerts

SET alerted\_at = NOW()

WHERE id = ?;



### **Acceptance criteria**

1. Endpoint exposed for the cron to call, which handles the alert emails being sent to specified recipients.
2. Cron created to call the new endpoint every 1 minute

Expose API endpoints to Create, Read and Update surge alert configuration data in the reply\_manager\_surge\_alert\_config table.

## **Endpoints**

POST /surge-alert/:brand\_id/config  
PUT /surge-alert/:brand\_id/config/:config\_id  
GET /surge-alert/:brand\_id/config

## **POST and PUT Request Body example:**

****

{

"surge\_reply\_count\_per\_period": 10,

"surge\_reply\_period\_in\_ms": 60000,

"alert\_cooldown\_period\_in\_ms": 900000,

"emails\_to\_notify": ["admin@brand.com", "team@brand.com"],

"enabled": true

}

## **GET Response**

Return an array of all the configs for the brand\_id in the route

Use the microservice-boilerplate to create this repo and service.  
Confirm if Signoz has been implemented in here.

Update README where necessary

Develop a stored procedure evaluateReplyManagerSurges to evaluate hidden tweet activity and trigger surge alerts.

## **DB Event**

Create a database event to **run the stored procedure** evaluateReplyManagerSurges **every 5 minutes**.

### **Event Details:**

* **Runs every 5 minutes**.
* Calls evaluateReplyManagerSurges (details below)
* Ensures alerts are **evaluated at a regular interval** without manual intervention.



CREATE EVENT evaluate\_surge\_alerts\_event

ON SCHEDULE EVERY 5 MINUTE

DO CALL evaluateReplyManagerSurges();

## **Stored Proc Logic:**

1. Fetch all active surge alert configurations from reply\_manager\_surge\_alert\_config.
2. For each configuration, fetch the count of hidden\_tweets where:
   1. owner\_id (retrieved from hidden\_tweets.widget\_id) matches brand\_id.
   2. hidden\_at falls within the last surge\_reply\_period\_in\_ms.
   3. The tweet is **not retro-hidden**.
3. If the count **≥ surge\_reply\_count\_per\_period**:
   1. Check if an alert has been sent in the last alert\_cooldown\_period\_in\_ms.
      1. If **NO alert has been sent**, create a new entry in reply\_manager\_surge\_alerts.
      2. If **YES**, do nothing
4. Always update last\_evaluated\_at in reply\_manager\_surge\_alert\_config after evalualting a config entry.

## **Acceptance Criteria:**

* Stored procedure evaluateReplyManagerSurges evaluates hidden tweet counts correctly.
* A new entry in reply\_manager\_surge\_alerts is created if a threshold is met.
* The stored procedure always updates last\_evaluated\_at for the config

Create a new table reply\_manager\_surge\_alert\_config to store configuration settings for surge alerts at the brand level. This table will store thresholds for detecting surges in hidden replies and notification settings.

### **Table: reply\_manager\_surge\_alert\_config**

Stores configuration settings for surge alerts per brand.

Add triggers for created\_at and updated\_at where required.

Query performance could be optimized by indexing hidden\_tweets.hidden\_at.



CREATE TABLE reply\_manager\_surge\_alert\_config (

id UUID PRIMARY KEY,

brand\_id UUID NOT NULL,

surge\_reply\_count\_per\_period INT NOT NULL,

surge\_reply\_period\_in\_ms INT NOT NULL,

alert\_cooldown\_period\_in\_ms INT,

emails\_to\_notify JSON NOT NULL,

enabled BOOLEAN NOT NULL DEFAULT TRUE,

last\_evaluated\_at TIMESTAMP NULL,

created\_at BIGINT NOT NULL,

created\_by varchar(36) NOT NULL,

updated\_at BIGINT NULL,

updated\_by varchar(36) NULL,

FOREIGN KEY (brand\_id) REFERENCES brands(id)

);

Create a table reply\_manager\_surge\_alerts to store surge alert records whenever a threshold is met.



CREATE TABLE reply\_manager\_surge\_alerts (

id UUID PRIMARY KEY,

surge\_alert\_config\_id UUID NOT NULL,

created\_at BIGINT NOT NULL,

surge\_amount INT NOT NULL,

alerted\_at TIMESTAMP NULL,

config\_snapshot JSON NOT NULL,

FOREIGN KEY (surge\_alert\_config\_id) REFERENCES reply\_manager\_surge\_alert\_config(id)

);

Enhance the front-end system by implementing a **Surge Alert** feature that allows **Ambassadors and Admins** to configure alerts for high activity in hidden replies. The surge alert configuration will apply **across all posts accumulatively** and only include posts that are already part of the existing setup for **hiding promo posts** for a brand.

#### **Requirements:**

1. **Permissions & Access:**
   1. The option to enable surge alerts should only be available to **Ambassadors and Admins**.
2. **Alert Configuration Options:**
   1. Allow users to **specify email addresses** to receive surge alerts.
   2. Allow users to define **X-number of hidden replies** within **Y-recent minutes** that will trigger a surge alert.
   3. Implement a **cooldown interval** (default: **15 minutes**) to prevent excessive notifications after an initial alert.
3. **Monitoring & Display:**
   1. Display current **throughput information** related to hidden replies (multiple format options).

#### **Acceptance Criteria:**

* Surge alert configuration can only be accessed by Ambassadors and Admins.
* Users can add email addresses to receive alerts.
* Users can configure **X-hidden replies over Y-recent minutes** as the threshold for alerts.
* Alerts respect the **15-minute cooldown** before triggering again.
* Current **throughput data** for hidden replies is visible on the front end.
* Surge alerts only apply to **posts included in the existing setup** for hiding promo posts.

#### **Notes:**

* Ensure that the **surge alert logic applies across all posts accumulatively** and follows the existing system's rules for hidden promo posts.