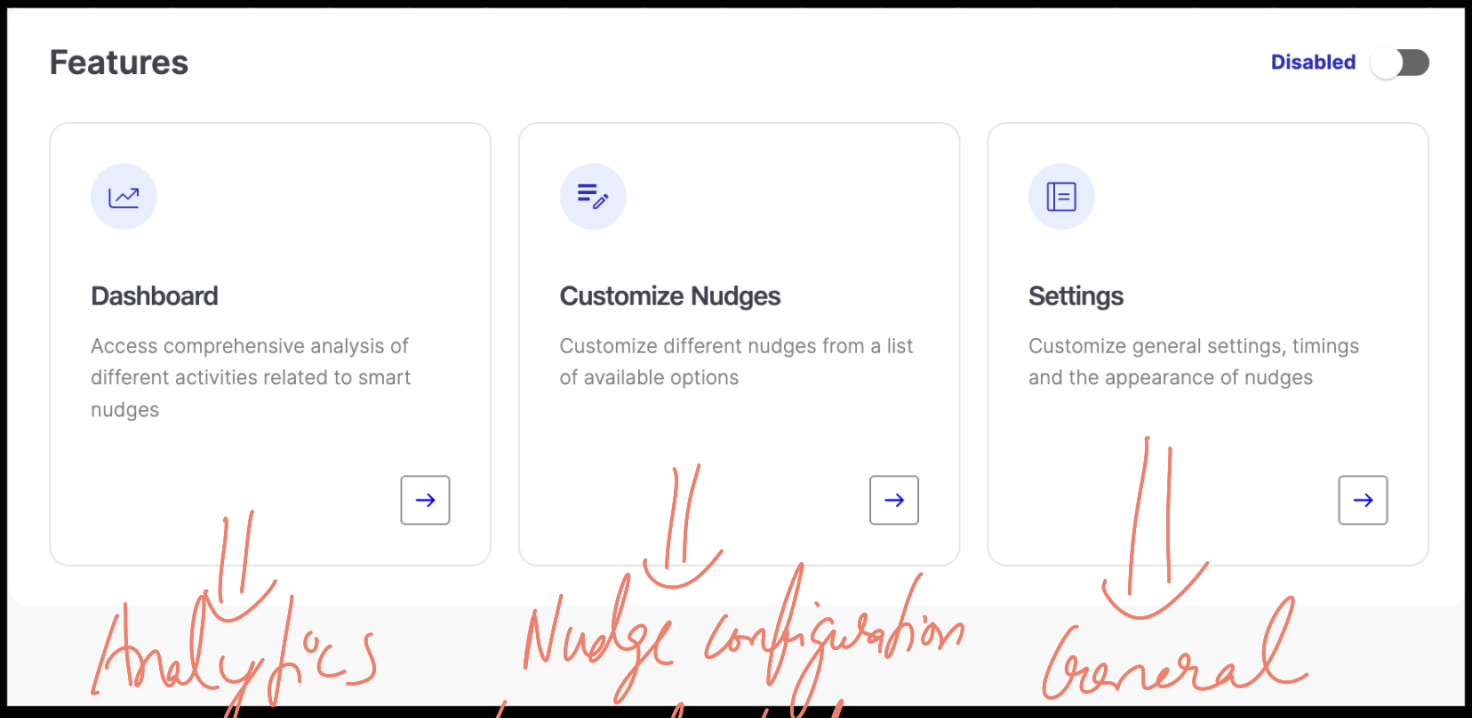
[Smart Sales Nudge System]

Functional Requirement

1. Website customer able to see Nudges for recent activity on website
2. Seller should be able to configure Nudge look and feel
3. Seller should be able to configure on which platform, timings and position of Nudge
4. Analytics for user activity on Nudge.
5. 4 broad category
   - Recent purchase
   - Popularity
   - Recent summary
   - Friction

Non functional

1. System should be highly available + low latency
2. Consistency may be compromised
3. Should be able to handle Million events per day
4. Secure + Durable



Add to Cart

Set up the appearance of add to cart smart nudge

Enabled

Settings

Header Tag
Turn ON toggle button to enable header tag

Notification Time
Turn ON toggle button to enable notification time

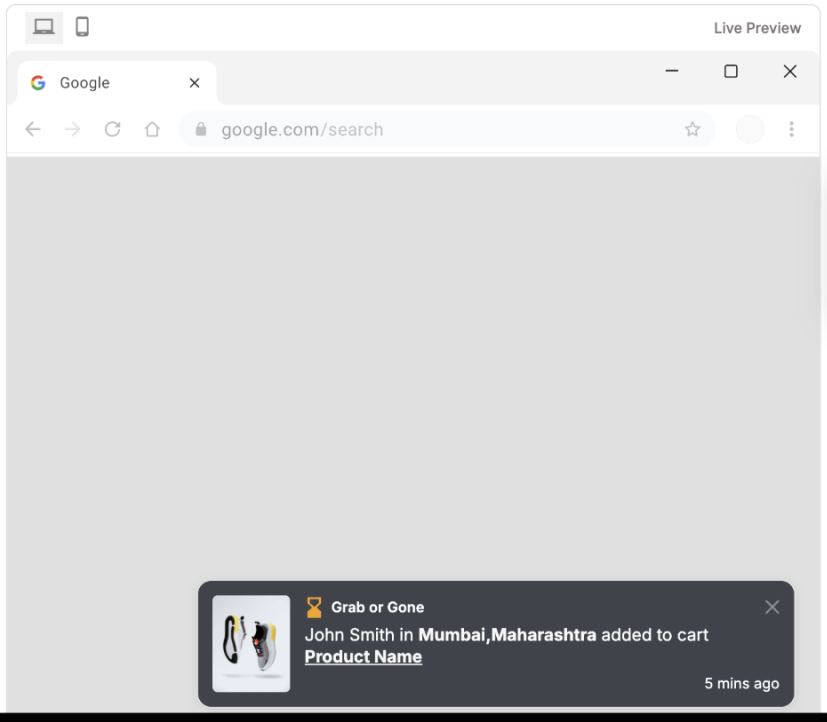
Anonymous
Turn ON toggle button to hide the name of customer

Select Pages
Choose page(s) on which this nudge will appear

Product Listing Page, Categories Page, Brand Listing Page, N...

Highlight Color
Select a color to highlight the emphasized text

Save



↳ Single Nudge config screen

Home > Mika > Features > General Settings

General Settings

Set up positions, timing, devices, and themes for smart nudges

Settings

Profile

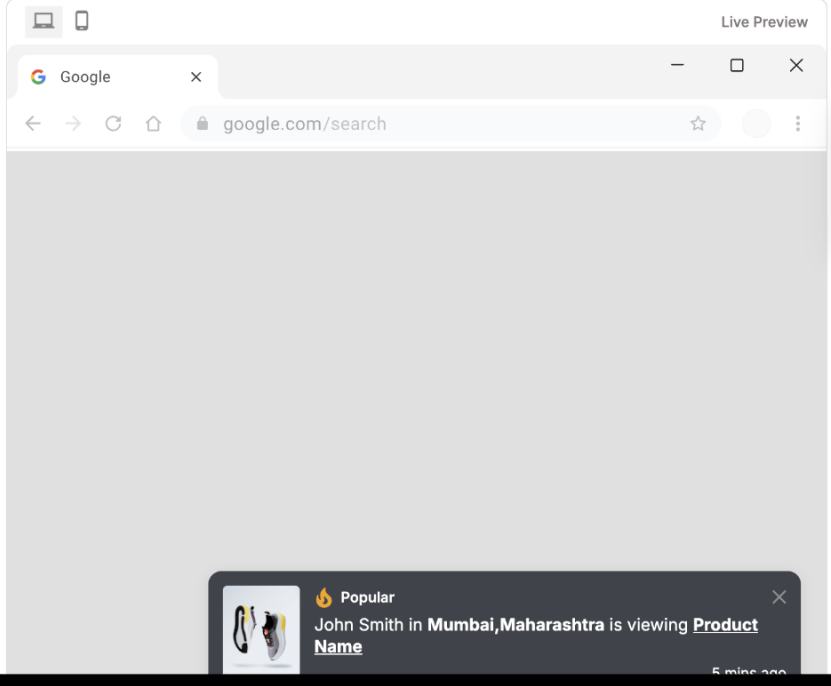
Positions
Configure the position of pops for desktop and mobile screen

Timing
Set up different levels of timings for visibility of smart nudge

Devices
Select the target platform(s)

Themes
Select light or dark theme for smart nudges

Save



↳ Common setting for all Nudges

Devices

Select the target platform(s)

- [x] Desktop
- [x] Android
- [x] iOS

Themes

Select light or dark theme for smart nudges

- ( ) Light
- (•) Dark

Positions

Configure the position of pops for desktop and mobile screen

Desktop	( ) Bottom Left	(•) Bottom Right
Mobile	( ) Top	(•) Bottom

Timing

Set up different levels of timings for visibility of smart nudge

Display initial smart nudge after a delay of

5s 10s 15s 20s

Delay of time interval between smart nudge

5s 10s 15s 20s

Max. number of notifications

5 10 15 20

Max. number of notifications per page

1 2 3 4

Types of Nudges

1. Product: View, Add to cart, Order
2. Order, cart summary
3. Live, Recent visitor
4. ** Product review
5. Low stock

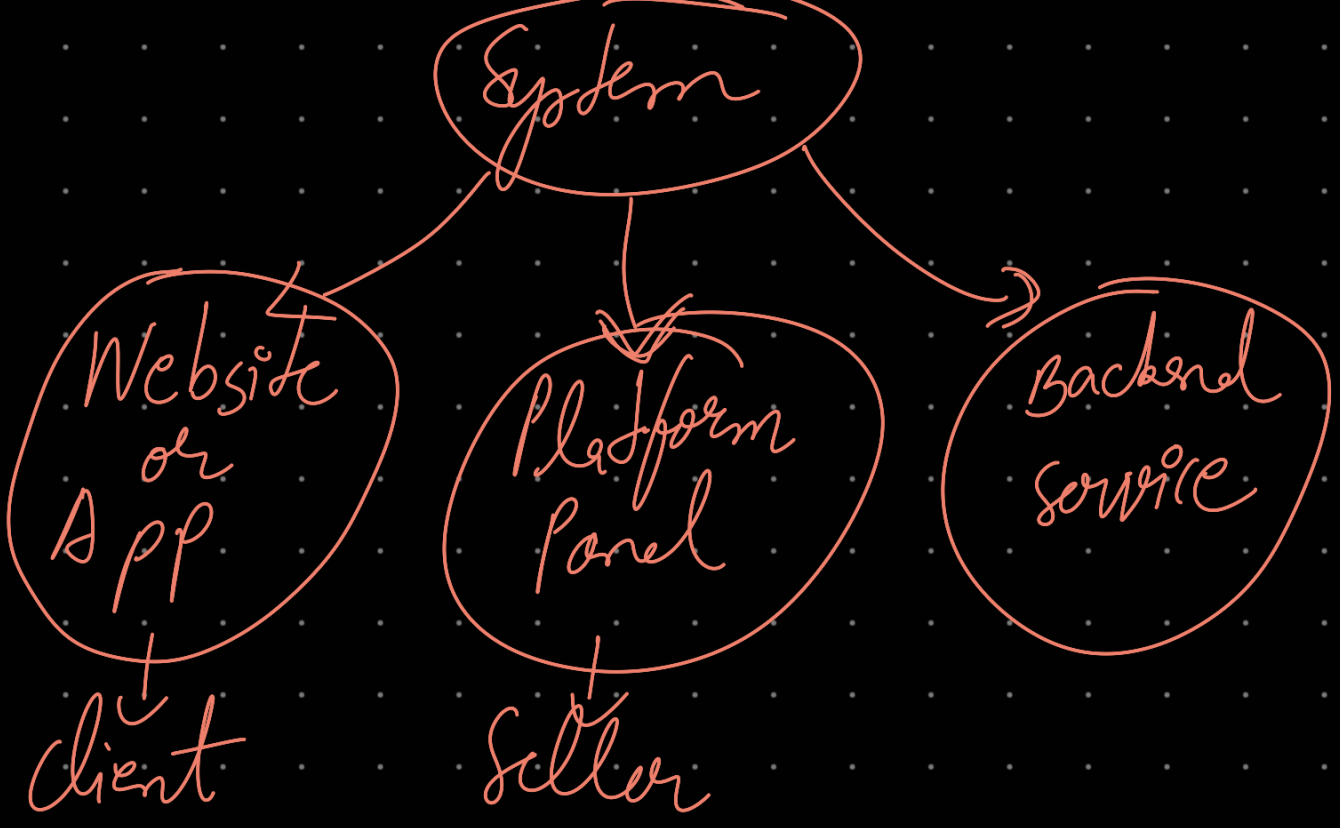
↳ Categorization for Backend service

[Add to cart, Review, Order] → Recent purchase

[View, Live visitor, Recent visitor] → Popularity

[Cart, Order Summary] → Recent summary

[Low stock] → Friction



[Capacity Estimation]

* 1.5 M/day → 45 million/month
  └) user visits
* Considering page view events
  = 1.5 M/day

Size of 1 event data ≈ 500 Bytes

DB requirement for 1 day = 1.5M * 500 ≈ 800 MB

For 1 month = 800 * 30

≈ 24 GB

Peak traffic 10x = 240 GB/month

[Database choice]

① Large writes

② No fixed schema for events and Nudges

③ No need of complex joins

④ Horizontally scalable

⑤ Availability is important than consistency

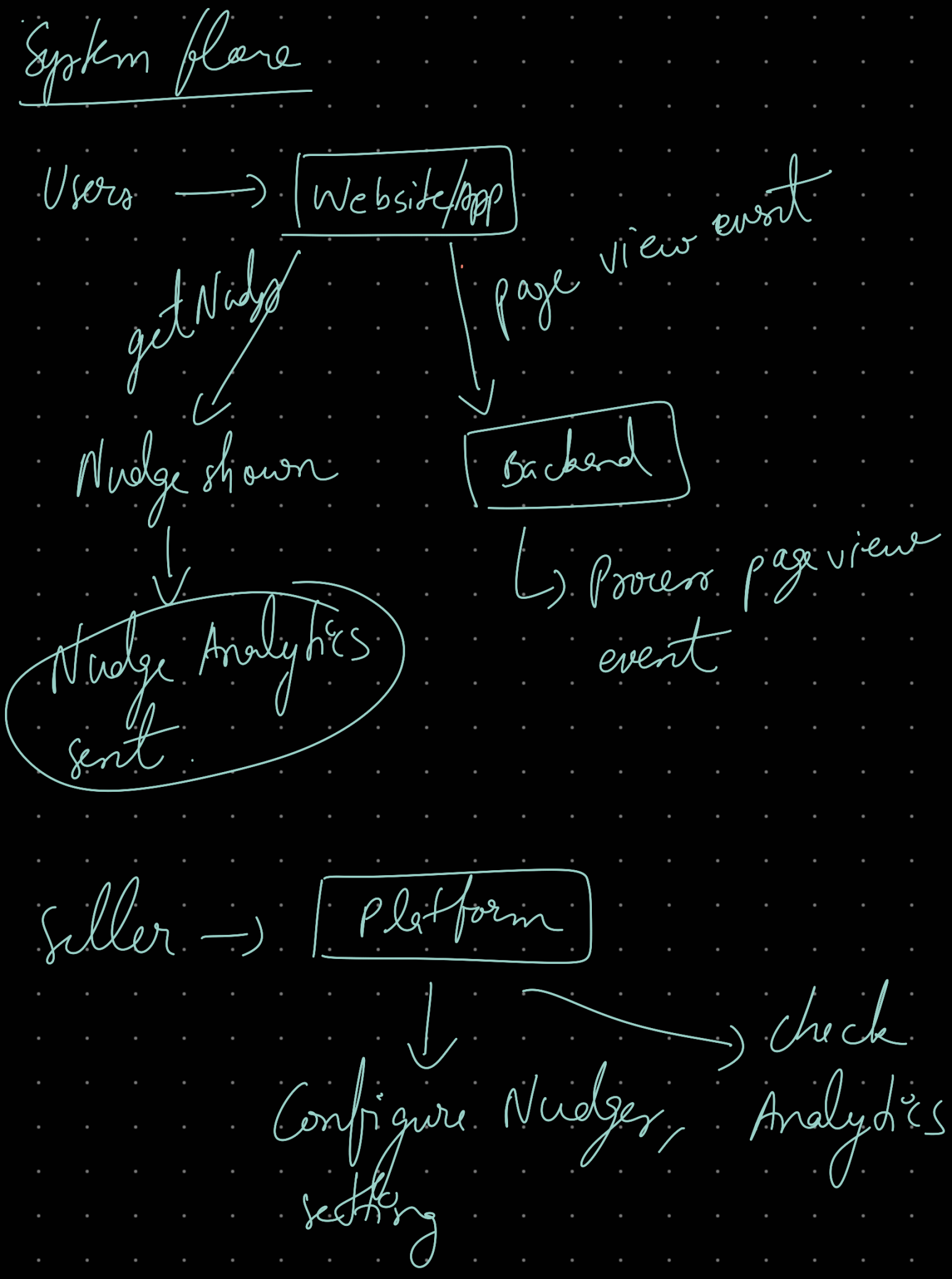
⇓

NO SQL

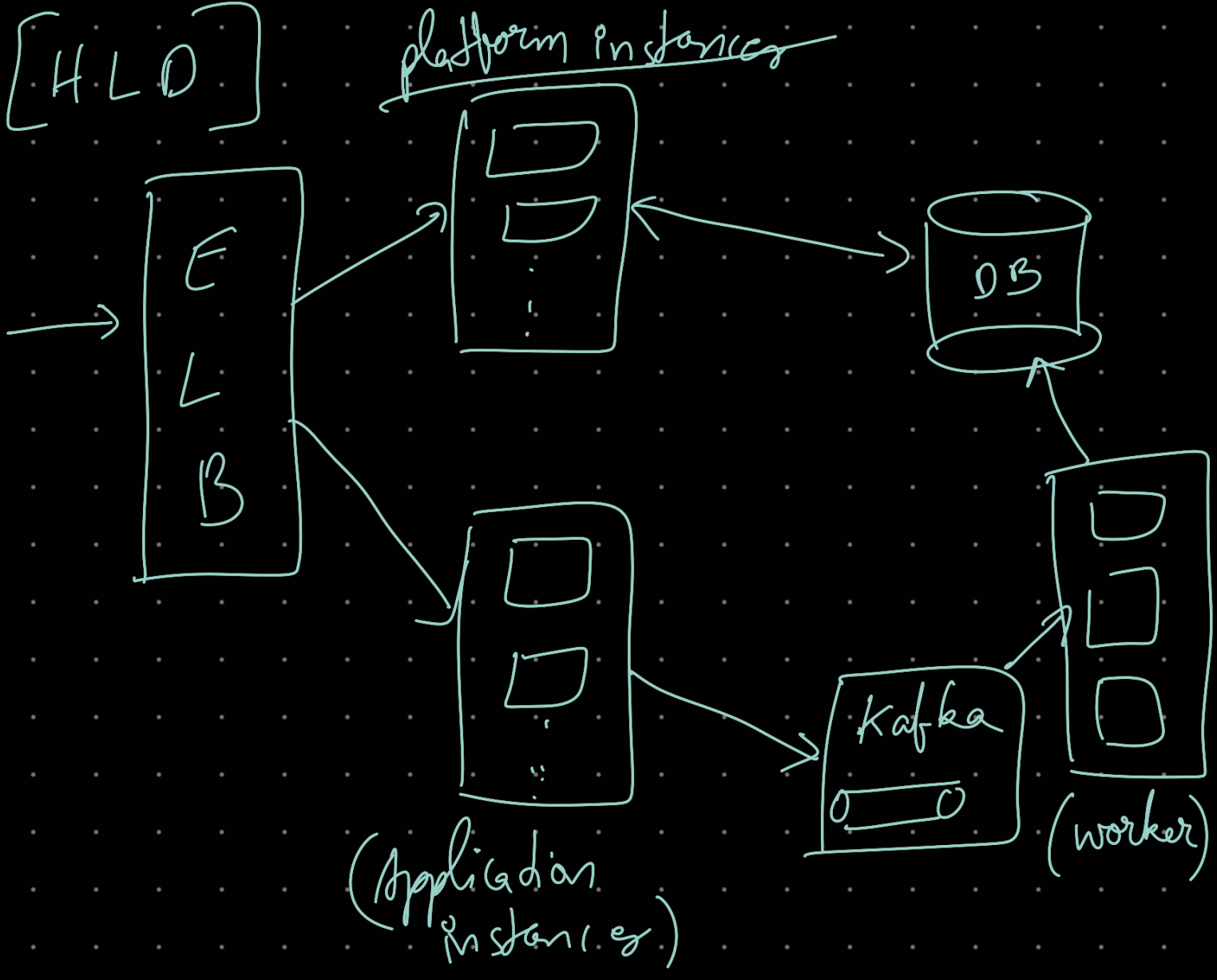
⇓

MongoDB

System flow


Users
Website/app
getNudge
page view event
Nudge shown
Backend
Nudge Analytics sent
Process page view event
Seller
Platform
Configure Nudges, settings
check Analytics

[HLD]



[Event processing flow]

Page view event

Application server

producer

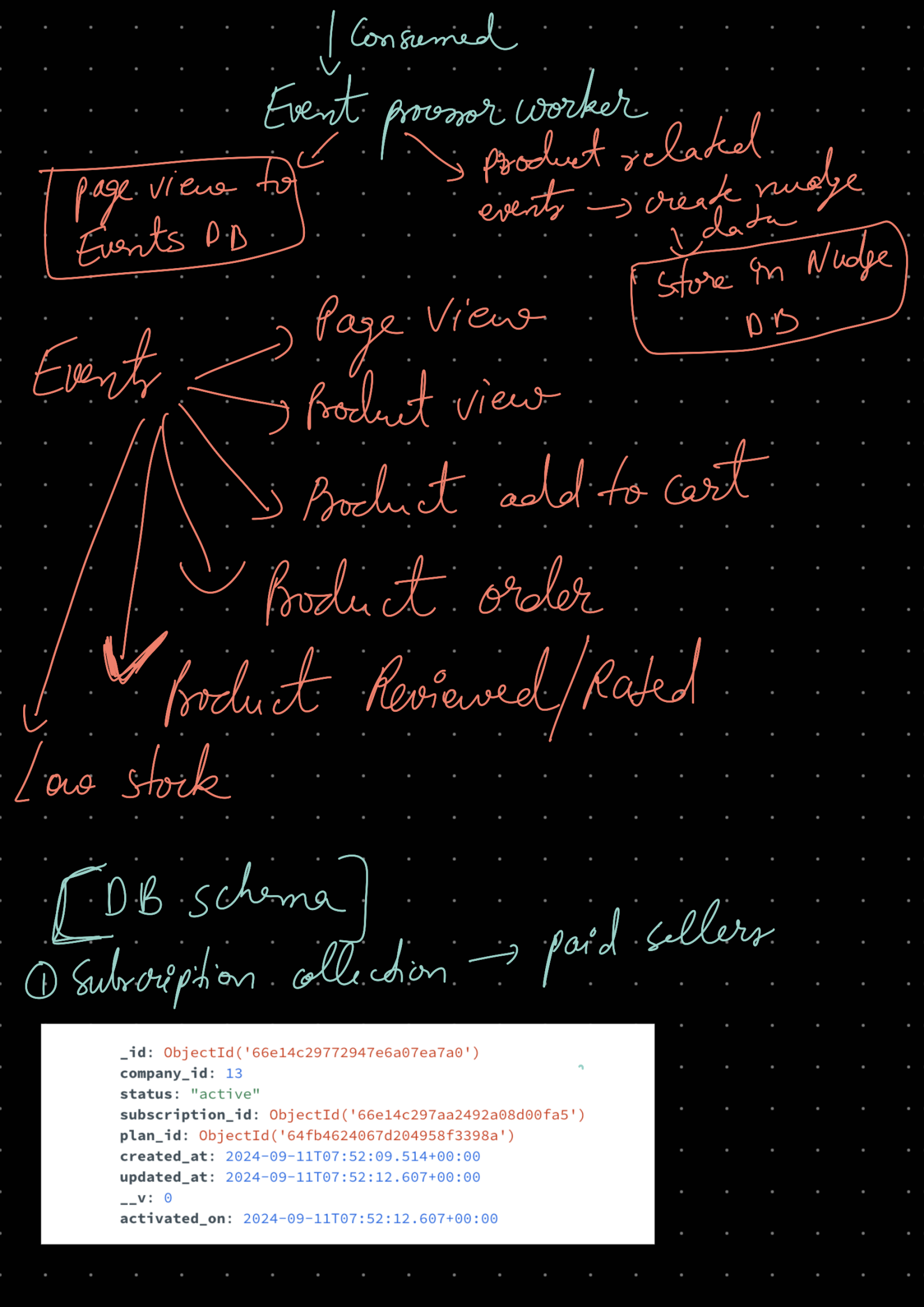
Event recieved from Review service

Kafka queue

For scalability

From Product

Stock service



[DB schema]

① Subscription collection → paid sellers

```
_id: ObjectId('66e14c29772947e6a07ea7a0')
company_id: 13
status: "active"
subscription_id: ObjectId('66e14c297aa2492a08d00fa5')
plan_id: ObjectId('64fb4624067d204958f3398a')
created_at: 2024-09-11T07:52:09.514+00:00
updated_at: 2024-09-11T07:52:12.607+00:00
__v: 0
activated_on: 2024-09-11T07:52:12.607+00:00
```

(2) Service enabled/disabled

```
_id: ObjectId('64fb4635067d204958f33998')
company_id: "1"
application_id: "6369fc8124ab1b2008130d8a"
extension_active: false
__v: 0
```

(3) Types of subscription plans

```
_id: ObjectId('64fb4624067d204958f3398a')
name: "Standard"
displayName: "Standard"
tagline: ""
is_active: true
▸ price: Object
▸ features: Array
yearly_plan: false
created_at: 2023-09-08T16:04:52.606+00:00
updated_at: 2023-09-08T16:04:52.606+00:00
__v: 0
```

(4) Nudge Configs

```
_id: ObjectId('64fb464a067d204958f339b3')
company_id: "1"
application_id: "6369fc8124ab1b2008130d8a"
is_active: false
type: "ADD_TO_CART"
[redacted]
[redacted]
▸ configs: Object
__v: 0
created_at: 2023-09-08T16:05:30.251+00:00
updated_at: 2024-03-05T09:12:24.453+00:00
```

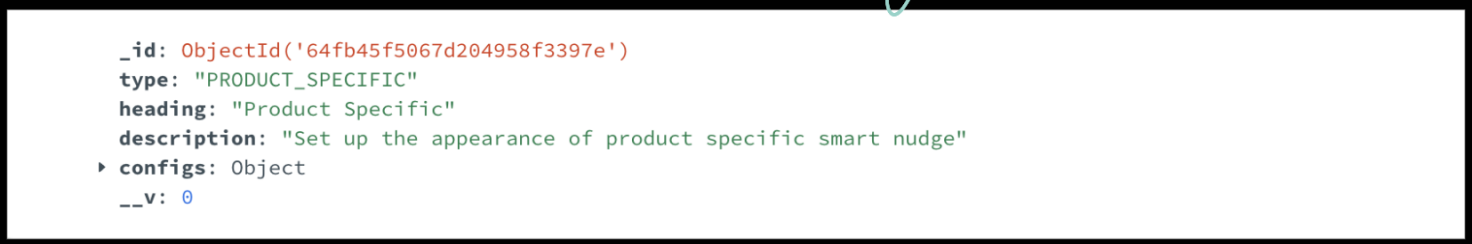
(5) General Nudge settings

```
_id: ObjectId('64fb4631067d204958f33995')
company_id: "1"
application_id: "6369fc8124ab1b2008130d8a"
▸ general_settings: Object
created_at: 2023-09-08T16:05:05.088+00:00
updated_at: 2023-09-13T04:28:42.052+00:00
__v: 0
```

(6) Events (Page Views)

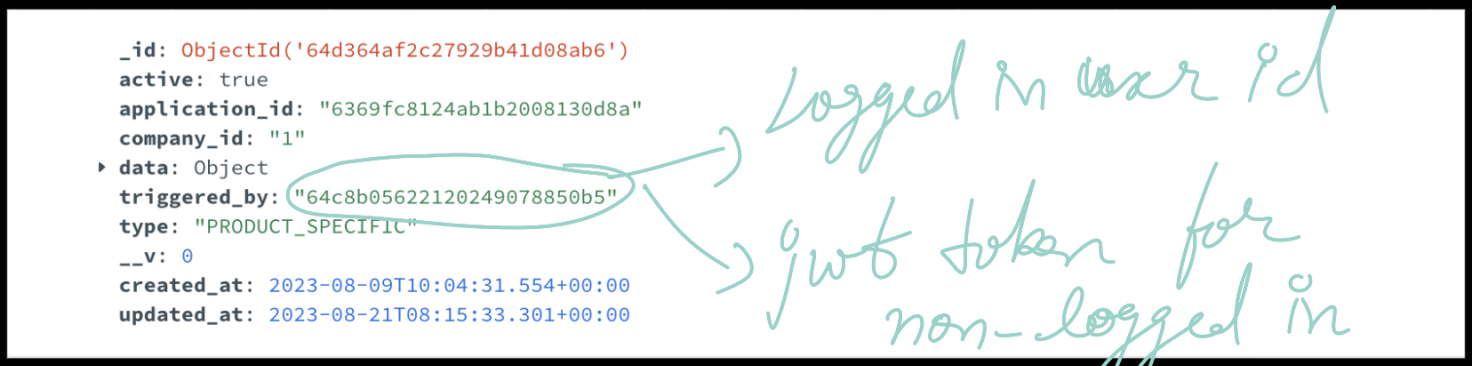
```
_id: ObjectId('64d364292c27929b41d00cb2')
active: true
application_id: "6369fc8124ab1b2008130d8a"
company_id: "1"
▾ data: Object
    screen_view: "home"
  ▸ user: Object
triggered_by: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTY5MTU3NTMzNzgyMSwiaWF0I…"
type: "VISITOR"
__v: 0
created_at: 2023-08-09T10:02:17.983+00:00
updated_at: 2023-08-09T10:02:17.983+00:00
```

(7) Default Nudge categories



```
_id: ObjectId('64fb45f5067d204958f3397e')
type: "PRODUCT_SPECIFIC"
heading: "Product Specific"
description: "Set up the appearance of product specific smart nudge"
▸ configs: Object
__v: 0
```

(8) Nudges collection



[Summary Nudge Design]

Approach 1 :- Run CRON Job

⇓

get addtocarts in Last time period

↳ Acc. to Nudge config provided by seller

E.g → Last Day

→ Last week

→ Last Month

⇓

Aggregate

||

Create Nudge object

||

Store in Nudge DB

Problems → What if we have 1 lakh document or 1 Million

||

CRON POD Memory (RAM) will overflow

How did we optimised this?

Approach 2: Batch processing

① Get data in batch size

② Aggregate

③ Repeat

Problem → Slow and Huge processing time for CRON, not scalable

How did we optimized this?

Approach 3: Bull queue + Workers (Multiple)

CRON Job Runs Every 5 minute

⇓

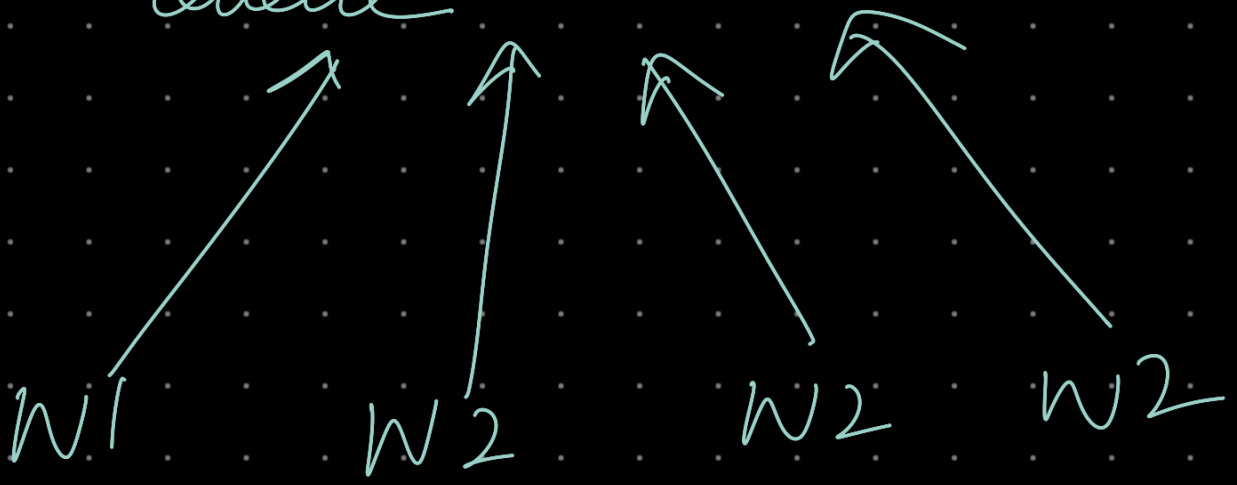
Creates (10,000) batch Job

⇓

Creates Job with batch Number + Batch Size + timeRange + type of Nudge

⇓

Job added to Redis Queue



* Each worker pores and do aggregation using Batch size

⇓

Maintain redis key by

(company id _ application Id _ Nudge type)

↓

=> Each worker Inc the count using its aggregated value

L) Redis INC is Atomic

⇓

Use "completed" event on Bullqueue for run logic on completion of each worker.

```
const redis = new Redis();

let totalJobs = 0; // Track total jobs
let completedJobs = 0; // Track completed jobs

// Create jobs and push to the queue
async function createJobs(batchSize, totalDocuments) {
  totalJobs = Math.ceil(totalDocuments / batchSize);
  for (let i = 0; i < totalJobs; i++) {
    await batchQueue.add({ batchNumber: i, batchSize });
  }
}

// Worker to process jobs
batchQueue.process(async (job) => {
  const { batchNumber, batchSize } = job.data;
  console.log(`Processing batch ${batchNumber}`);

  // Simulate processing and increment Redis counters
  const counts = { ADD_TO_CART: 200, ORDER: 150 }; // Replace with real aggregation logic
  await redis.incrby("ADD_TO_CART", counts.ADD_TO_CART);
  await redis.incrby("ORDER", counts.ORDER);

  return `Batch ${batchNumber} processed`;
});

// Track job completion
batchQueue.on("completed", async (job) => {
  completedJobs++;

  console.log(`Job ${job.id} completed. ${completedJobs}/${totalJobs} done.`);

  // Check if all jobs are completed
  if (completedJobs === totalJobs) {
    console.log("All jobs completed. Aggregating final results...");
    await aggregateAndSaveResults();
  }
});
```

worker code

Save to Nudge DB

[Event Archiving]

* Using AWS S3 storage as cold storage to archive data older than 2 calendar Months

CRON Jobs Runs 1st of every Month at Midnight

||

Creates Job with batch size and timestamp in (Redis)

Eg → 1 week = 1 batch

Multiple parallel worker instances

||

=> Uploads to S3 with Date Range in CSV format

L> This approach is scalable

=> Create one more collection to store Metadata like S3 url, Date range etc.

[Analytics Module Design]

[Schema]

Sample Schema (MongoDB example):

json

Copy code

```
{
  "nudgeId": "nudge123",        // Unique Nudge identifier
  "eventType": "viewed",         // "viewed", "clicked", or "closed"
  "timestamp": ISODate("2024-12-09T10:00:00Z"), // Timestamp of interaction
  "userId": "user123",           // Optional, if tracking individual users
  "sessionId": "session123"      // Optional, for tracking sessions
}
```

Frontend

⇓

event sent to server

⇓

pushes to Analytics Queue in Redis

⇓

Aggregate into local array

Consider 1000 batch

⇓

Once array reaches 1000 length

→ Bulkwrite to DB → reset array

→ RUN CRON to delete this DB ('Documents before 30 day from Current date)

[API Design]

① Platform/Seller Panel API

/platform/v1/

- GET /settings/:app_id
- POST /setting/:app-id
- PUT /setting/:app-id

↳ General setting

GET /nudge-config/:app-id/:type

PUT /nudge-config/:app-id/:type

↳ Nudge config

GET /analytics/:app_id

↳ Analytics

---

(2) Website / App API

GET /session → to create sessionID in case of Non logged in user

GET /target-device

↳ To check iOS, Android, Web

GET /nudges → List of Nudges

↳

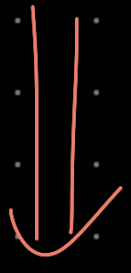
query -> page, count, alreadyShownIds, productDetails
↳ optional

header -> Jwt token -> Non logged In
user id -> Logged In

↳ So that user should not see nudges created by his own activity.

POST /nudge-analytics

↳ schema defined above



[Challenges and Scalability]

① Handling high volume events
  ↳ Message queue

② After 1 month, Query performance degraded → reason high volume data
  ↳ ① Indexing
  ↳ ② Archiving older events
    ↳ Cost saving
  ↳ ③ Read Replica for MongoDB
  ↳ ④ Caching Analytics for some duration