

Riding Score and Statistics

Idea

Aim is to gamify parts of app to bring in more user engagement. Assigning a score to each day/ride can drive retention and daily check-ins by users.

Logic

Riding Score is based on 4 key driving behaviours-

- 1. Harsh Acceleration
- 2. Sudden Braking
- 3. Sharp Turning
- 4. Over-speeding

Ajjas Lite devices can be set-up to send automatic alarms for each of these events, whereas we have added logics in `multinodeg1ep` to detect such behaviour for Ajjas Pro devices.

Max Riding Score is 100 Points. We calculate a penalty based on the count of occurrences for the above four defined behaviours. Then, this penalty is scaled according to the travelled distance. Distance was chosen as the scaling factor as it is easier to scale across different hardware types.

Event Name	Multiplier
Harsh Acceleration	-3
Sudden Braking	-4
Sharp Turning	-3
Over-speeding	-3

NOTE: These multipliers are decided based on testing data.

Formulae

Distance is scaled differently below and above 50 KM. Detailed mathematical interpretations can be found [here](#).

```

function getRideScoreParameters(distance) {
  if (distance ≤ 6) return { base: 1.7, scale: 0.233, floor: 3 };
  if (distance ≤ 10) return { base: 1.0, scale: 0.075, floor: 6 };
  if (distance ≤ 25) return { base: 0.7, scale: 0.020, floor: 10 };
  if (distance ≤ 50) return { base: 0.4, scale: 0.012, floor: 25 };

  if (distance > 50) {
    const currSlab = Math.ceil(distance / 50) - 1;
    const prevScale = reduceNumber(0.27, 13, currSlab - 1);
    const currScale = reduceNumber(0.27, 13, currSlab);

    return { base: prevScale, scale: (prevScale - currScale) / 50, floor: 50 * currSlab };
  }
}

function getRideScoreFactor(distance) {
  if (distance ≤ 3) return 1.7;

  const { base, scale, floor } = getRideScoreParameters(distance);
  return base - (scale * (distance - floor));
}

```

Code for Calculating Score Factor

Note that the distance passed should be in KMs.

Logic is to define slabs for different distance ranges

For Distances below 50 KM

Base Scales defined for each slab are constants and not derived mathematically.

Range	Base Scale	Per KM Scale
0 - 3 KMs	1.7	-
3.1 - 6 KMs	1.0	0.233
6.1 - 10 KMs	0.7	0.075
10.1 - 25 KMs	0.4	0.020
25.1 - 50 KMs	0.1	0.012

For Distances above 50 KM

Base Scale starts at 0.27 and decreases by a constant 13% each 50 KM. For example,

Range	Base Scale	Per KM Scale
1 - 50	0.27	-
51 - 100	0.235	0.000700
101 - 150	0.204	0.000620
151 - 200	0.178	0.000520
201 - 250	0.155	0.000460

Per KM Scale is derived by subtracting the base for current slab from base of previous slab and then dividing it by the number of kilometres in the current slab.

Hence, to derive scale for any distance x ,

Previous Slab Base Scale - (Distance Deviated from Start of Current Slab * Per KM Scale of Current Slab)

Example,

For 99 KM

Previous Slab $\rightarrow 1 - 50 \rightarrow 0.27$

Current Slab $\rightarrow 51 - 100 \rightarrow 0.235$

Deviation from Start of Slab $\rightarrow 99 - 50 \rightarrow 49$

Per KM Scale $\rightarrow (0.27 - 0.235) / 50 \rightarrow 0.0007$

\rightarrow Scaling Factor $= 0.27 - (49 * 0.0007)$

Now, to get the final score of a ride,

1. Add the sum of products for each negative driving event count with its multiplier.
2. Multiply it with the scaling factor calculated above.
3. Subtract it from the max riding score (100)

```
function getRideScore(distance, behaviourCounts = {}) {
  const factor = getRideScoreFactor(distance / 1000);

  // allocate score penalty based on predefined points
  let weightedPenalty = 0;
  weightedPenalty += (behaviourCounts.OVERSPEED || 0) * RIDE_SCORE_POINTS.OVERSPEED;
  weightedPenalty += (behaviourCounts.HARSH_ACCELERATION || 0) * RIDE_SCORE_POINTS.HARSH_ACCELERATION;
  weightedPenalty += (behaviourCounts.SHARP_TURN || 0) * RIDE_SCORE_POINTS.SHARP_TURN;
  weightedPenalty += (behaviourCounts.SUDDEN_BRAKE || 0) * RIDE_SCORE_POINTS.SUDDEN_BRAKE;

  let score = MAX_RIDE_SCORE - (Math.abs(weightedPenalty) * factor);
  score = Math.min(MAX_RIDE_SCORE, score);
  score = Math.max(MIN_RIDE_SCORE, score);

  return score;
}
```

Function for Calculating Final Riding Score

Database Schemas

For Riding Score, we store 2 keys in `dailyStats` - `rideScore` and `rideScoreCounts`.

These keys are included in each `dailyStats` document for Ajjas Pro and Lite devices, as well as each element in the `rides` array for each document.

`rideScore` \rightarrow number; denotes ride score for the day between 25 and 100

`rideScoreCounts` \rightarrow { OVERSPEED: 0, HARSH_ACCELERATION: 0, SHARP_TURN: 0, SUDDEN_BRAKE: 0 }

Driving events for both Pro and Lite are stored in a common mongo collection named `drivingEvents`.

```
1 {
2   typ: 1 | 2 | 3 | 4,
3   uid: number, vid: number,
4   did: number, hwtyp: 1 | 3,
5   lat: number, lng: number,
6   ts: number,
7 }
```

Note that the timestamp (`ts`) is saved in seconds.

Front-end

To keep the logic for Riding Score same across platforms and prevent bugs in the future, we decided to not calculate the score on the front-end. Hence, riding score in app is shown from the `dailyStats` collection in MongoDB which is updated every day through the `timelineDailyStats` cron job.

Logic to Sync Statistics in Realm w/ Server

MobileApp stores `dailyStats` in the JavaScript Realm Schema `DailyRides`.

Local Sync TS (`statsMdt`) is stored with each Vehicle in the `Bikes` collection. This is the timestamp (`ms`) when DailyStats for this vehicle were last synced w/ the server. Hence, the latest stat stored in `DailyRides` will have this `ts`.

Server Sync TS is received by the app through Dynamic data, each time the app is opened (`new WS connection`). This dynamic data contains the timestamp when `dailyStats` were last updated in Mongo against each vehicle's ID owned by the user.

If the app finds that its local sync ts is outdated, it fetches data from app's local sync ts till the current time. In case local ts is not present in the app, a fetch request with `mdt = 0` is sent to the server and the server responds with last 2 months' `dailyStats` data.

Daily Ride Threshold

Since the data in `dailyStats` is calculated through a cron job, we cannot rely on it to show stats for the current day (or last day if cron job has not run yet). To work around this limitation, mobile app calculates its own stats from 2 days ago in real-time using timeline events.

Since the app never calculates riding score on its own, it calls a separate API if it needs to show the riding score within the daily ride threshold.

`/gl/users/timeline/getRidingScore`

query parameters:

`vid` → vehicle id of user's vehicle

`hwtyp` → since timeline events are stored separately in Scylla for Pro and Lite

`dst` → necessary for calculating the scaling factor

`sdt` -> optional; day start ts for which riding score needs to be calculated; by default, it's today's sdt

The API combines driving events fetched from -

- `drivingEvents` collection in Mongo, if sdt is before yesterday

- t1e in Scylla, if sdt is after yesterday

This behaviour is to account for driving events generated using saved data.

CRM

Additionally, Riding Score can be seen on CRM.

1. From dailyStats - in user's timeline daily rides

Start Date
06/01/2024

End Date
06/07/2024

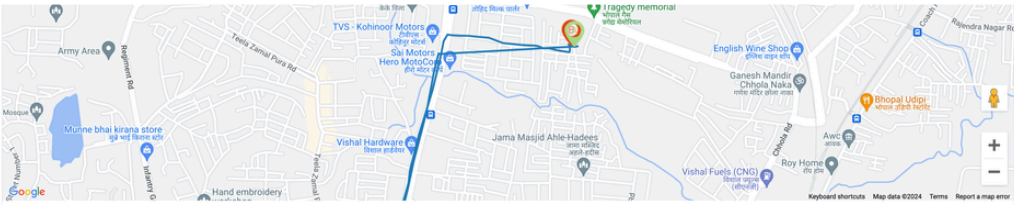
Submit

Export to Sheet

Vehicle Id	Date	Total Distance (km)	Total Duration (hr:min)	Average speed	Top speed	Riding Score	Ping Count
83428	7 Jun 2024	27.09	1:14	21.88	77.00	96 Overspeed: 0 Sharp Turn: 0 Sudden Brake: 2 Harsh Acceleration: 1	Ignition On: 787 Ignition Off: 11 Motion Off: 798 Power Off: 798 Ignition On With Motion Off: 787 Speed Dist: 0-4: 5 5-10: 116 11-20: 311 21-30: 200 31+: 155 Speed Dist: Ignition Off With Motion Off: 11 Sleep count: 11 Average Sleep Duration: 1853.9090909091 Max Sleep Duration: 20141 Angle Zero: 798 Temperature Distribution: Total: 798
83428	5 Jun 2024	21.59	0:60	21.62	97.00	97 Overspeed: 0 Sharp Turn: 0 Sudden Brake: 1 Harsh Acceleration: 1	Ignition On: 635 Ignition Off: 7 Motion Off: 642 Power Off: 642 Ignition On With Motion Off: 635 Speed Dist: 0-4: 3 5-10: 147 11-20: 210 21-30: 129 31+: 146 Speed Dist: Ignition Off With Motion Off: 7 Sleep count: 7 Average Sleep Duration: 2896.4285714285716 Max Sleep Duration: 20112 Angle Zero: 642 Temperature Distribution: Total: 642

crm.ajjas.com/tlDailyRides

2. Calculated on CRM - by fetching vehicle's driving events



Ride Score		
Event	Count	Multiplier
Overspeed	0	-3
Sharp Turn	0	-3
Sudden Brake	2	-4
Harsh Acceleration	1	-3
Total		-
Scaling Factor: 0.3746		
Final Score: 95.88		

crm.ajjas.com/timelineRideLogs