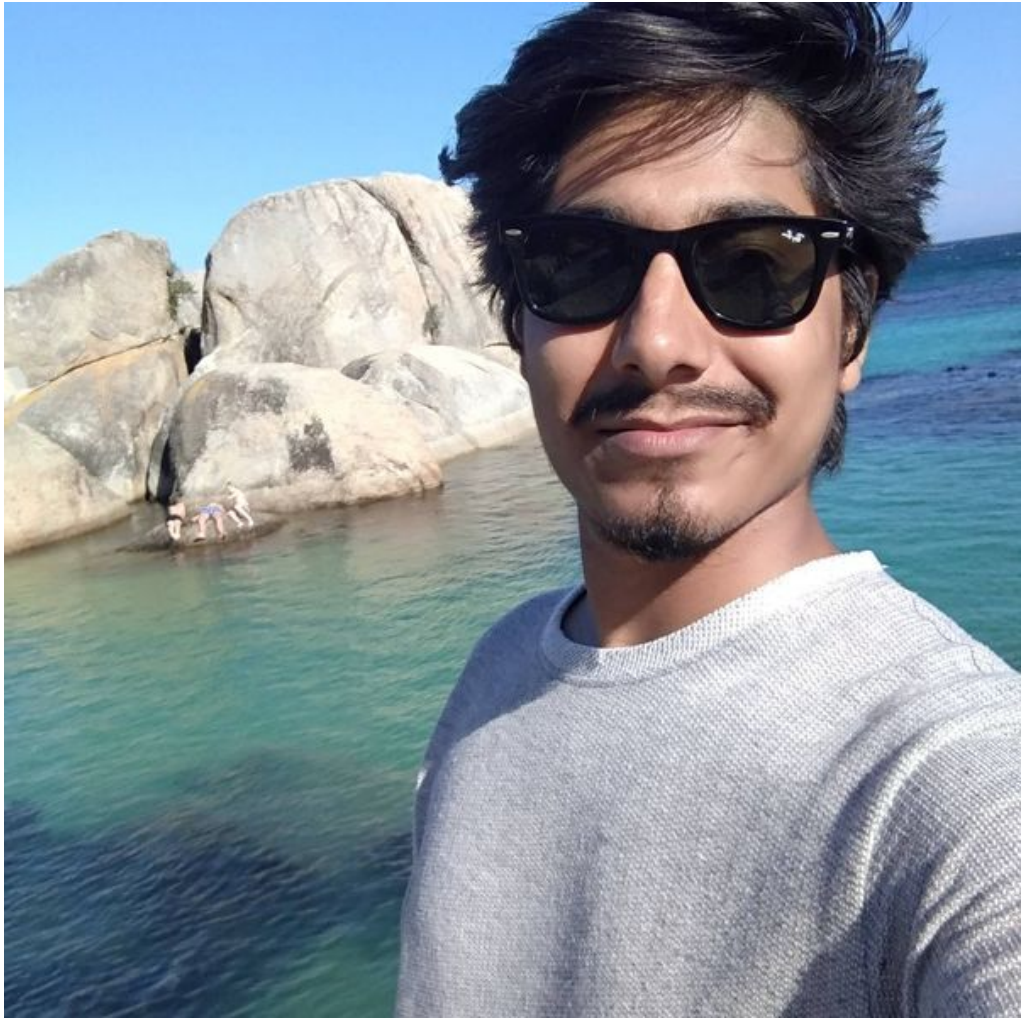
Performance

How we at OnceHub reduced our public API response time by **86%**

About me

- I work as a Senior Software Engineer at CardUp.
- For me, it’s been around 4 years working in the industry.
- I live in Delhi, India. We are know for food. 😊
- Incoming Masters student in Distributed Systems Engineering at TU Dresden.
- Apart from coding: Football, travel and learning German.

/in/raounak-sharma/

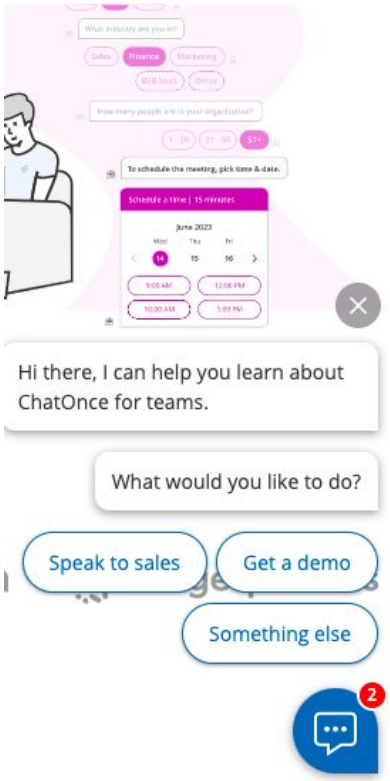


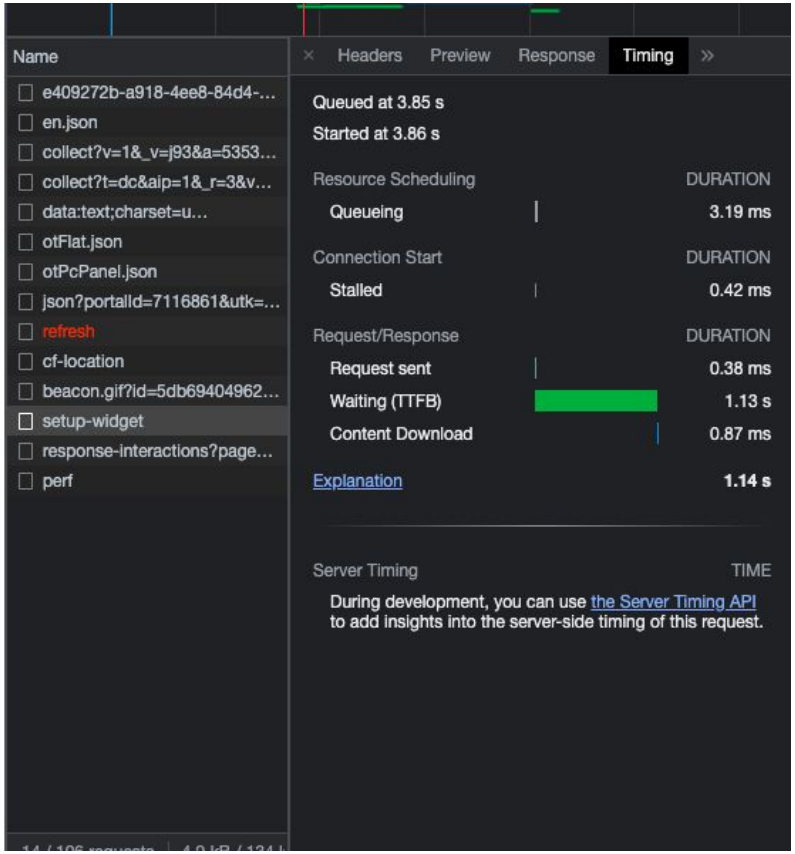
The protagonist

From 10,000 feet bird view definition:

It’s a highly customizable chatbot integrated with different software products, which help in scheduling meeting with your customers.

oncehub.com/chatonce





OnceHub’s API endeavour from 5 secs to 1 sec

Late 2019

- The product “ChatOnce” was in initial phase.
- We were building features upon features and everything was going beautifully.

Note:

- We were not that ignorant of best practices. We had SOLID and good config on multiple environments.

Mid 2020

Problems arise:

- It was getting hard to build new features
- Every feature adding significant time in the API

Reasons:

- The new features coming were computationally very heavy.
- Time is always a constraint.

No Product launch

Reason:

The API response time reached to 4.5 to 5 seconds.

The trade-offs does not favour the launch.

Effect:

Need to change the route to Refactoring.

Diagnosing

Where to start from 🤷‍♂️

Profiling

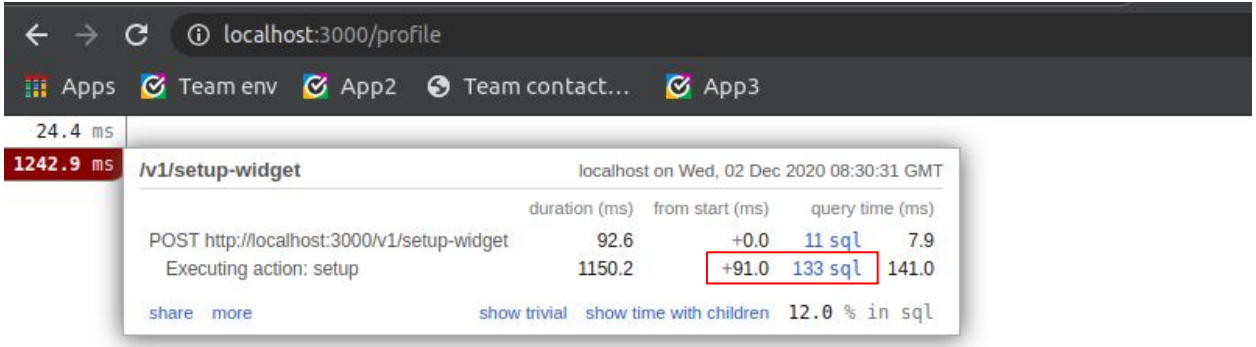
It is a form of dynamic program analysis that measures, for example, the space (memory) or time complexity of a program, the usage of particular instructions, or the frequency and duration of function calls.

The biggest tool we have when it comes to finding bottleneck of performance issues.

StackProf (Sampling profiler)

Rack mini profiler (Tracing profiler)

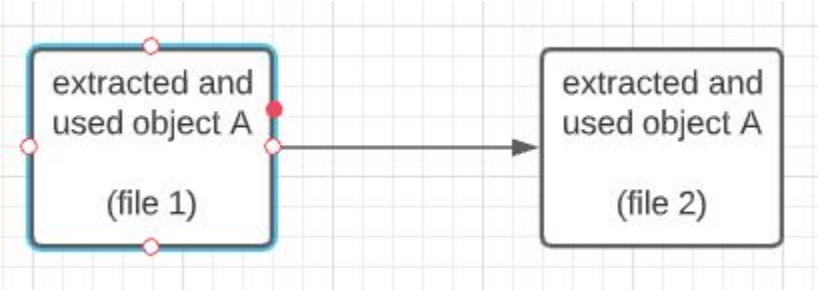
Rack mini profiler



Problem 1: Too many SQLs

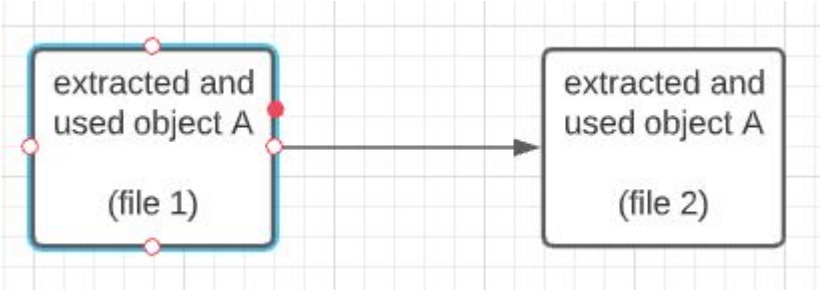
Reducing the number of DB calls

- We were extracting out same objects multiple times from <u>different files</u>.



Reducing the number of DB calls

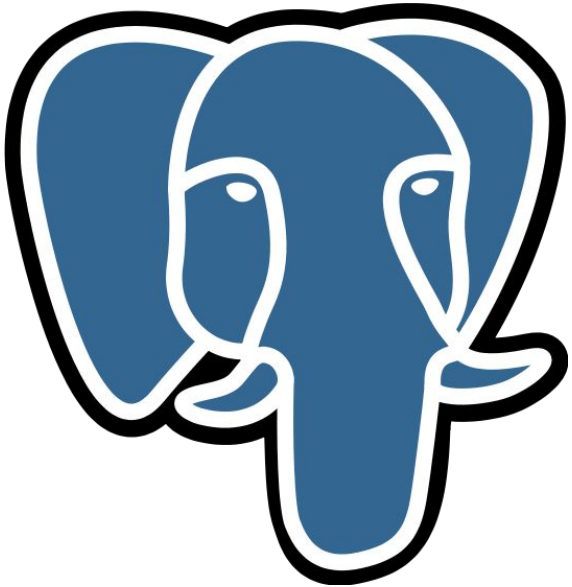
- We were extracting out same objects multiple times from <u>different files</u>.
- Solution: Services Objects, thread local variables (only in thread safe environment)



StackProf (Sampling profiling)

```
% stackprof --text tmp/stackprof-cpu-myapp.dump
==================================
  Mode: wall(1000)
  Samples: 27368 (12.42% miss rate)
  GC: 450 (1.64%)
==================================
     TOTAL    (pct)     SAMPLES    (pct)     FRAME
      9836  (35.9%)        9836  (35.9%)     PG::Connection#async_exec
      5350  (19.5%)        5350  (19.5%)     PG::Connection#exec_params
      3794  (13.9%)        3794  (13.9%)     PG::Connection#exec_prepared
      3234  (11.8%)        3234  (11.8%)     PG::Connection#exec
      2907  (10.6%)        2907  (10.6%)     PG::Connection#prepare
       490   (1.8%)         489   (1.8%)     #<Module:0x0000561715f939b8>.connect
       391   (1.4%)         391   (1.4%)     ActiveRecord::ConnectionAdapters::ConnectionPool::Reaper#run
       354   (1.3%)         354   (1.3%)     (marking)
       221   (0.8%)         221   (0.8%)     ActiveModel::AttributeMethods::ClassMethods#define_proxy_call
        94   (0.3%)          94   (0.3%)     (sweeping)
        34   (0.1%)          34   (0.1%)     ActiveRecord::AttributeMethods::ClassMethods#method_defined_within?
        34   (0.1%)          27   (0.1%)     ActiveRecord::AttributeMethods::Read::ClassMethods#define_method_attribute
        23   (0.1%)          23   (0.1%)     Concurrent::Collection::NonConcurrentMapBackend#get_or_default
        41   (0.1%)          22   (0.1%)     ActiveRecord::DynamicMatchers#respond_to_missing?
        19   (0.1%)          19   (0.1%)     Module#method_visibility
        45   (0.2%)          17   (0.1%)     Module#redefine_method
        16   (0.1%)          16   (0.1%)     block (4 levels) in class_attribute
        16   (0.1%)          14   (0.1%)     ActiveSupport::Inflector#apply_inflections
        14   (0.1%)          14   (0.1%)     Concurrent::Collection::NonConcurrentMapBackend#[]
        14   (0.1%)          14   (0.1%)     ActiveRecord::Associations::Builder::Association.define_readers
        13   (0.0%)          13   (0.0%)     #<Module:0x00007fc4844216c0>.storage_to_output
        13   (0.0%)          13   (0.0%)     Module#silence_redefinition_of_method
        15   (0.1%)          13   (0.0%)     ActiveRecord::AttributeMethods::Write::ClassMethods#define_method_attribute=
        12   (0.0%)          12   (0.0%)     ActiveRecord::DynamicMatchers::Method.pattern
        11   (0.0%)          11   (0.0%)     ActiveModel::AttributeMethods::ClassMethods::AttributeMethodMatcher#method_name
        10   (0.0%)          10   (0.0%)     ActiveRecord::Delegation::DelegateCache#generate_relation_method
        35   (0.1%)           9   (0.0%)     Class#class_attribute
        10   (0.0%)           8   (0.0%)     Array#extract_options!
         8   (0.0%)           8   (0.0%)     ActiveRecord::ConnectionAdapters::AbstractAdapter#type_map
         8   (0.0%)           8   (0.0%)     #<Module:0x00007fc48432d610>.set_name_cache
```

Observing sudden spike



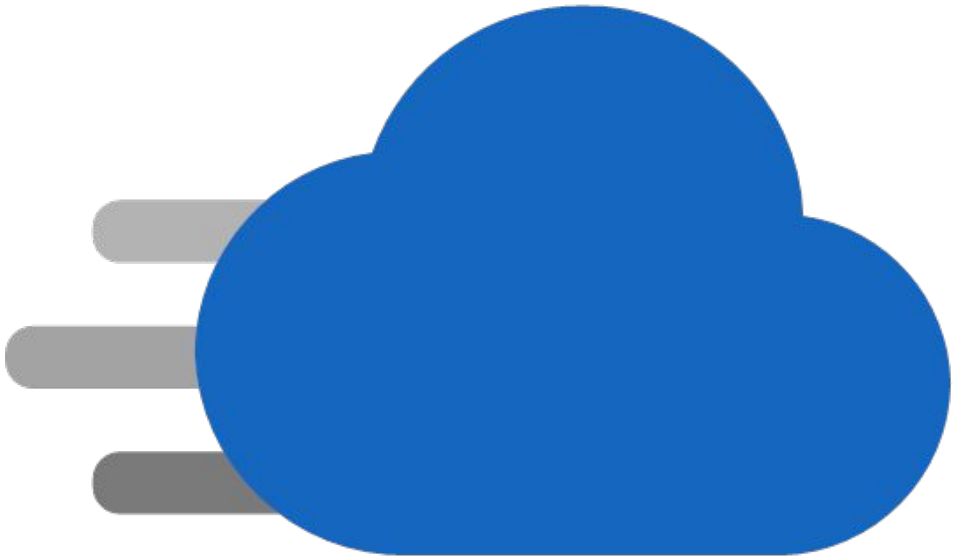
```
=========
Value: [[1]]
Time: 0.003929965
================

Log: Audience toggle in setup call: true
Log: New request ChatOnce engine
=========
Value: [[1]]
Time: 3.311013932
================

Log: Audience toggle in setup call: true
Log: New request ChatOnce engine
Log: New request ChatOnce engine
=========
Value: [[1]]
Time: 0.004276389
================
```

Usual latency between ChatOnce-engine and azure postgres DBMS

Problem 2: Latency in infracture

Latency in CDN

- Around 250ms was added by Azure CDN
- Servers were in Europe and we were accessing it in India
- Ofcourse cost and time are to be considered.



Removed latency

- Cause: Remote DBMS connection
- Opening a database connection is an expensive operation, especially if the database is remote
- Connection pool size.

- What worked for us:
  - Tweek DBMS settings (increase pool size if possible)
  - Keep the DBMS close
  - 6 * n = 3*n (server) + 2*n (RabbitMQ worker) + 1*n (background job)

Unnecessary API calls

Alternative

```
mavericksslogs_CL
| project requestId_g, serviceTag_s, path_s, TimeGenerated, method_s, requestId_s
| where requestId_s contains "6ahsjrb69de6"    ## Put eventID here
| order by TimeGenerated desc
| limit 20
```

Use this query and see how many requests to contact service has been made during the processing of that event.

Example: For eventId: `a0s5dcaafs7` It made only one request to contact service despite updating the contact as well as having contact-based routing

Completed. Showing results from the last 24 hours.

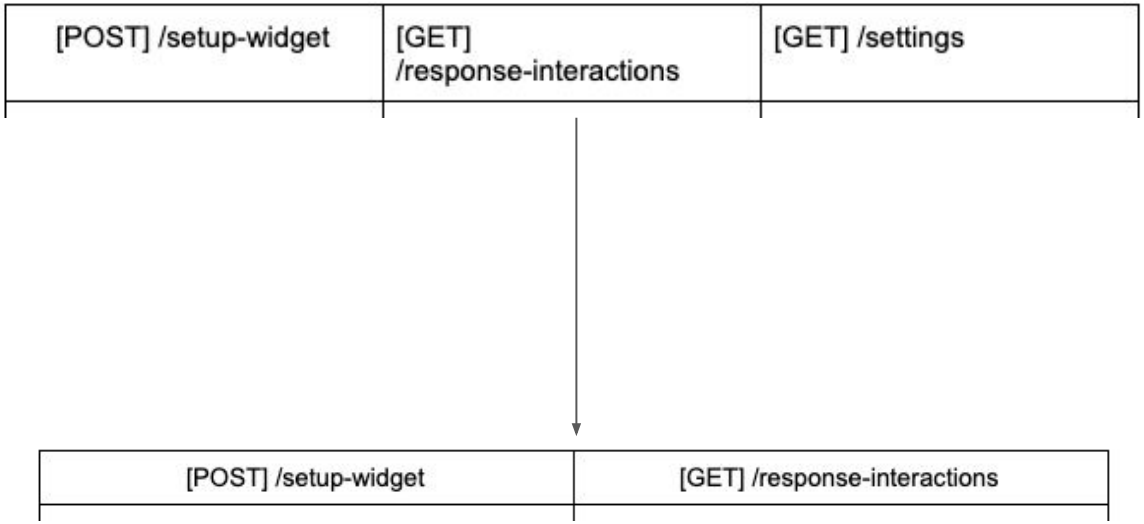
TimeGenerated [UTC]	requestId_g	serviceTag_s	path_s	method_s	requestId_s
11/9/2020, 6:05:57.984 AM		oh-field-service	/v1/account-field-validate/112294	POST	a0s5dcaafs7
11/9/2020, 6:05:49.937 AM		oh-event-logging	/v1/tag-history	GET	a0s5dcaafs7
11/9/2020, 6:05:49.864 AM		oh-contact	/v1/contact/CTC-Y0M5QXRM78	PUT	a0s5dcaafs7
11/9/2020, 6:05:49.769 AM		oh-event-logging	/v1/event	POST	a0s5dcaafs7

image-20201109-060627.png

Removing external calls

- Cause: Too much single responsibility
- Went from making 3 request to 2 request

- Made 500ms to 700ms difference



Removing internal API calls

- Causes:
  - Extra GET requests
  - POST request not returning updated data, so needed to make a GET request again.

```
==========
Caching Contact: {"isPreview"=>false, "isSoftDeleted"=>f
, "_id"=>"5fa268064c378e0a567134d7", "accountId"=>112294
-11-04T08:36:22.688Z", "timezone"=>"271", "city"=>"San A
tates"}], "state"=>[{"id"=>"OPT-74EK03VMWP", "value"=>"T
04T08:36:22.786Z", "updatedAt"=>"2020-11-04T08:36:22.786
==========


----------
Checking cache for contact: CTC-WLX8R8L3KM

Using contact cache for contact id: CTC-WLX8R8L3KM
----------


----------
Checking cache for contact: CTC-WLX8R8L3KM

Using contact cache for contact id: CTC-WLX8R8L3KM
----------
```

Reducing communication between services

Alternative

```
mavericksslogs_CL
| project requestId_g, serviceTag_s, path_s, TimeGenerated, method_s, requestId_s
| where requestId_s contains "6ahsjrb69de6"    ## Put eventID here
| order by TimeGenerated desc
| limit 20
```

Use this query and see how many requests to contact service has been made during the processing of that event.

Example: For eventId: `a0s5dcaafs7` It made only one request to contact service despite updating the contact as well as having contact-based routing

Completed. Showing results from the last 24 hours.

TimeGenerated [UTC]	requestId_g	serviceTag_s	path_s	method_s	requestId_s
11/9/2020, 6:05:57.984 AM		oh-field-service	/v1/account-field-validate/112294	POST	a0s5dcaafs7
11/9/2020, 6:05:49.937 AM		oh-event-logging	/v1/tag-history	GET	a0s5dcaafs7
11/9/2020, 6:05:49.864 AM		oh-contact	/v1/contact/CTC-Y0M5QXRM78	PUT	a0s5dcaafs7
11/9/2020, 6:05:49.769 AM		oh-event-logging	/v1/event	POST	a0s5dcaafs7

image-20201109-060627.png

Potholes to avoid

- Redundant data in API responses
  - It cause unnecessary computation
  - Infer behaviour from the object.

    Ex: {isMultiple: true}

- Ask only what is required
  - Remove backend computational part not just key from response

We have reached

Around 700ms to 900ms

Some stats

https://scheduleonce.atlassian.net/browse/ONCEHUB-33239

Jira Software | Your work | Projects | Filters | Dashboards | People | Apps | Create

Projects / OnceHub / ONCEHUB-32620 / ONCEHUB-33239

Raounak Sharma November 5, 2020, 9:57 AM Edited

Production stats

========== The following stats are for **new visitor** (when everything start from scratch)

`setup-widget` API timings, tested on Yola site

Clear cookies (New visitor - starting from scratch)	Before story (average time in seconds)	After story (average time in seconds)
Bot 1	5.06	1.14
Bot 2	4.93	1.26
Bot 3	4.23	1.15

`response-interaction` API timings, tested on Yola site

Clear cookies (New visitor - starting from scratch)	Before story (average time in seconds)	After story (average time in seconds)
Bot 1	1.01	0.37
Bot 2	1.53	0.38
Bot 3	1.16	0.33

Rele

Assigne

Reporte

Develop

Labels

Agile tea

Accepta

- On
  anc
  unc
- Rec

Effort

Story Po

Due date

========== The following stats are for **page refresh**

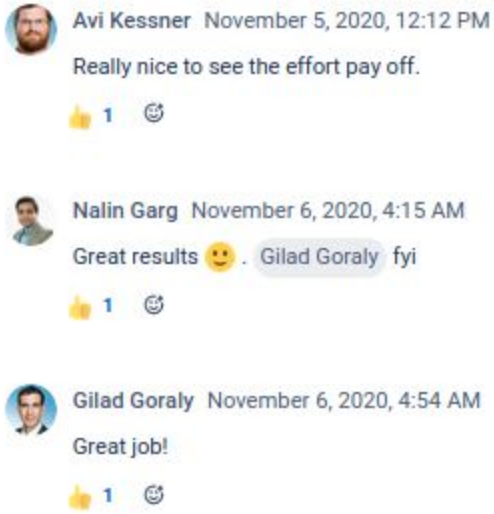
`setup-widget` API timings tested on Yola site

Page Refresh	Before story (average time in seconds)	After story (average time in seconds)
Bot 1	2.41	0.60
Bot 2	2.60	0.61
Bot 3	2.17	0.71

`response-interaction` API timings, tested on Yola site

Clear cookies (New visitor - starting from scratch)	Before story (average time in seconds)	After story (average time in seconds)
Bot 1	0.82	0.37
Bot 2	0.95	0.33
Bot 3	0.89	0.45

Everyone was happy



Obstacles we overcome & takeaways

- High number of database calls. (100+ SQL / request) ✅
- Unnecessary API calls. (internal + external) ✅
- Redundant data in API responses causing unnecessary computation ✅
- Ask only what is required ✅
- Latency in the infracture ✅

- Mostly it’s database
- Software design to reduce communication
- Think about the infracture

Thank you so much 🙇

Contact

raounaqsharma@gmail.com

@Raounaksharma

/in/raounak-sharma/