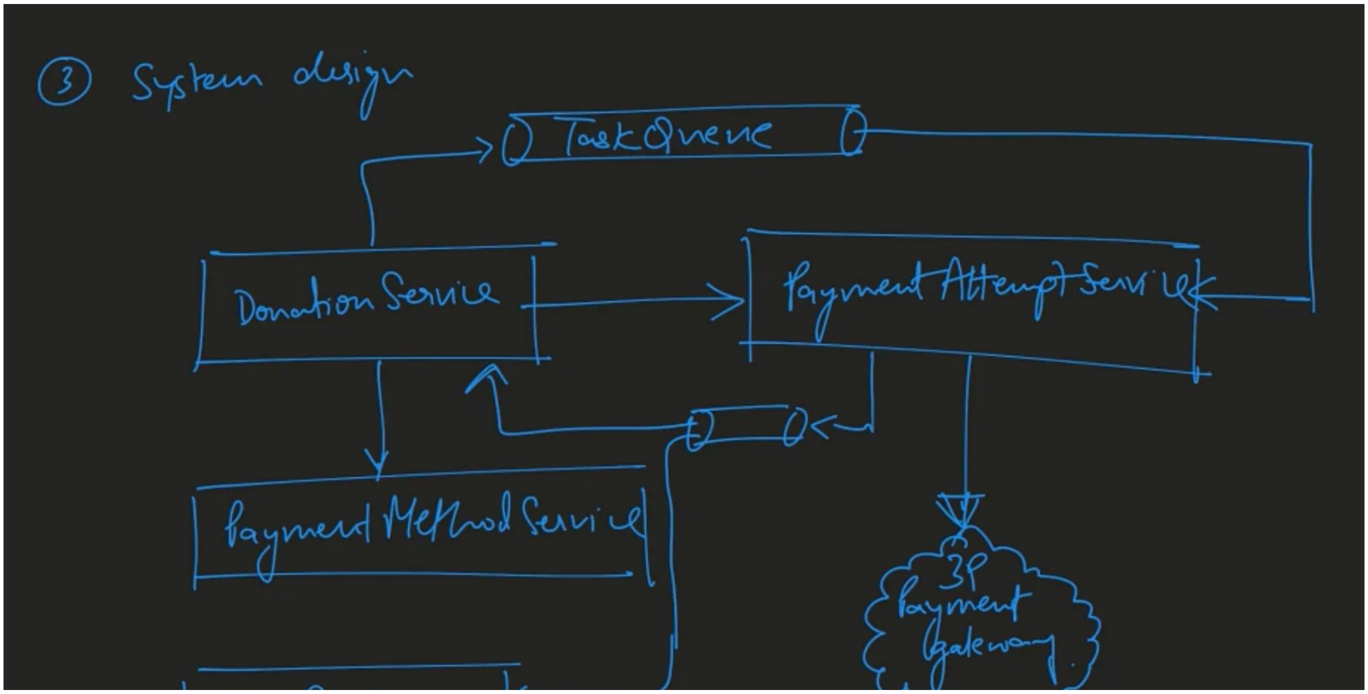
✦ Member-only story

System design for dummies — part 3 (Design a donation app)

S. G. · Follow

4 min read · Feb 7, 2022

72



This is the final part of a 4 part series on cracking the system design interview. Part 0 provided an overview of this article series, part 1 focused on the content you need to master to nail the interview and part 2 focused on the format you should follow during your system design interview.

In this final part I'll provide a real life example of a system design problem — *designing a donation app*. I lifted this problem from this discussion on Leetcode. Here is the problem statement:

- *You are required to design a donation app for a 3 day charity event across the US where you are expecting 3 million donations.*
- *You should accept details like customer name, credit card details etc.*
- *No pay specific knowledge is required (presume that you'd integrate with a 3rd party pay provider like Stripe or Braintree).*

If you are a visual learner like me & prefer videos to words, you can skip ahead to watching the video (it's listed at the bottom of this post) and then come back to the article. I do recommend reading through the entire article as it covers some key concepts that I didn't cover in the video.

Understanding requirements/goals

Focus on understanding the requirements to establish the bounds of the problem. Focus on the functional requirements (these would typically be specified in a PRD) as well as the non functional requirements.

For functional requirements, some of the questions you can ask are — *Will the users be logged in? Will they be using an app? What's the user experience if the payment fails or succeeds? Can the user view their past donations?*

> *Note that if you are a senior candidate, you are expected to own the problem & it's preferable that you come up with your own assumptions rather than expecting the interviewer to define constraints for you.*

For this problem, I've chosen to focus on donations and largely ignored how details about the charity are surfaced in the app.

The non-functional requirements will typically help establish the bounds/constraints of the problem. Use back of the envelope calculations to get the RPS (requests per second). 3 million donations over 3 days translates to 10 RPS. The shape of the traffic should be a consideration — e.g. most of the traffic for the donation app will come during the day time, and there might be further spikes in case of celebrity endorsements. For scale, we should build a system that can handle 100 RPS. To wrap up this section, you should comment on availability (improved via redundancy), latency (not super important here), throughput (improved via partitioning), consistency (eventual, read after write).

Data modeling & APIs

The main entities are User, PaymentMethod, Donation, PaymentAttempt which represent different micro-services (i've ignored Charity to reduce the scope). Draw out the entities & establish the relationship between them.

Note that Donation & PaymentAttempt could be tables within a single micro-service which would allow us to leverage on the ACID properties of relational databases & provide us with transactionality. However, it's common to follow the single responsibility principle & have the Donation service be separate from the PaymentAttempt service. This is justifiable if the PaymentAttempt service would be called by services other than the Donation service (e.g. a Shopping service). If we create a PaymentAttempt micro-service, it would be important not to leak donation related information into the PaymentAttempt service. For e.g. the donation_id would be renamed to something more generic like auxiliary_id (the id is called the donation_id in the video and needs to be corrected).

The biggest challenge in micro-service architecture is eventual state consistency. For e.g. if the payment attempt succeeds, we'd need to notify the Donation service so that the status of the Donation entry can be updated. If the webhook/event never reaches the Donation service, we risk being in an inconsistent state.

System architecture (component diagrams)

I recommend calling the 3rd party payment provider when the user is "in-session" (in-session means that the user is online & attempting to make the donation live). In the happy path the client would receive a status of 200 with donation status SUCCEEDED in real time. However, to design a system that is robust, we should enqueue retries if the 3rd party provider is unavailable & we receive a transient failure response. Retries can happen offline & we'd need to keep the overall state of the donation in sync with the status of the latest payment attempt.

Failure scenarios

Failure can happen at multiple points. Network blips can cause timeouts or databases can go down. We should also monitor for data inconsistencies. We should also rely on idempotency to ensure that a single donation does not result in multiple payment attempts.

Leader nodes can fail & one of the replicas will need to be promoted to be primary. Scenarios like split-brain should be prevented and replication lag should be monitored for.

Scaling

Partitioning can increase throughput. Partitioning by hash of the user_id is better than partitioning by range as range based partitioning can lead to more hot spots (e.g. imagine that a celebrity tweets and thousands of new users create accounts and start donating — they will all hit the same partition if user_ids are generated in an increasing order).

Video

Donation app for a 3 day charity event

Hope that you’ve enjoyed the article series & the video (sorry for the scratchy background noise — unfortunately it’s a part of the app I used to record the video).

If you have questions about the solution I presented or if you have suggestions on how this design could be improved please feel free to leave a question or comment. If you found this article useful, please consider following me on medium or reposting this article on social media. You may also be interested in reading my article called the LeetCode phenomenon.

System Design Interview
Programming
Software Development
Software Engineering

Written by S. G.

1K Followers

Follow

I write about programming, people management, interviews or anything else that I’m obsessing about. 12+ yes of experience across big tech and some in academia.

More from S. G.

SENIOR SUPREME TECH LEAD AT FLOORDASH. I GRADUATED FROM STANFORD 2 YEARS AGO
COOL BEANS
WHAT HAVE YOU BEEN UPTO ?
DISTRIBUTED STORAGE SYSTEM WHICH IS CONSISTENT, AVAILABLE, AND PARTITION TOLERANT.
IN MY SPARE TIME I ALSO PROVED THAT P= NP. THAT ONE WAS A TOUGHIE!
OK...
CAN YOU SWAP 2 VARIABLES WITHOUT USING A THIRD IN 30 SECONDS?
CAN I TAKE 5 MINS TO THINK IT THROUGH ?
THIS WAS A VERY HARD DECISION BUT WE WON'T BE PROCEEDING WITH YOUR CANDIDACY THIS TIME. PLEASE TRY AGAIN IN 6 MONTHS.
FUCK YOU

S. G.

The LeetCode Phenomena

In the winter of 2015, I was working as a Software Engineer at Microsoft and looking…

✦ · 6 min read · Nov 9, 2020

1.9K 11

S. G.

Be Agile, not Prolific

Agile interviewing structures preparation in sprints, with reflection at the core.

✦ · 8 min read · Dec 24, 2020

706 1

S. G.

System design for dummies—Part 0

If the system design interview has always been your achilles heel, this article is for you.…

✦ · 3 min read · Dec 23, 2021

S. G.

System design for dummies—part 2

Please read part 0 to get an overview of what to expect in this article series. Part 1 lists out…

✦ · 8 min read · Jan 1, 2022

67

88

See all from S. G.

Recommended from Medium

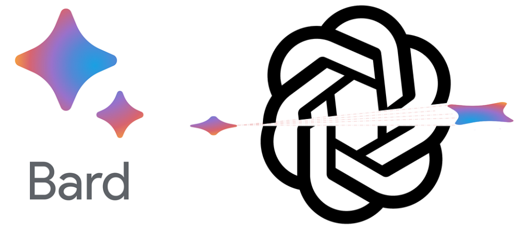
Full Stack from Full-Stack

Top 25 Kafka interview questions

1. What is Apache Kafka and why is it used?

5 min read · Aug 27

15



AL Anany

The ChatGPT Hype Is Over — Now Watch How Google Will Kill…

It never happens instantly. The business game is longer than you know.

· 6 min read · Sep 1

12.5K 388

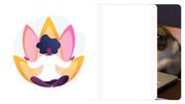
Lists

General Coding Knowledge

20 stories · 414 saves

It's never too late or early to start something

15 stories · 155 saves



Stories to Help You Grow as a Software Developer

19 stories · 442 saves

Coding & Development

11 stories · 205 saves

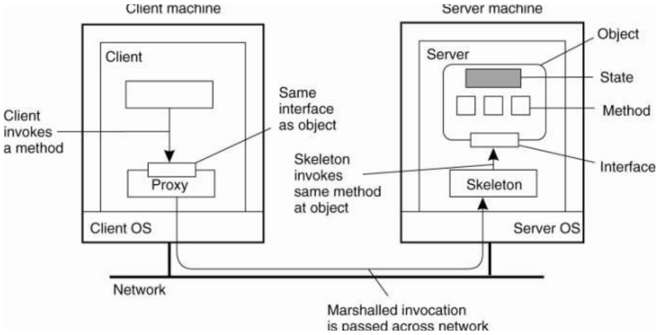
Dinkey mahawar

Meesho SDE3 Interview Experience

I interviewd for Meesho SDE3 Position in July'23. The process had 3 three rounds—...

2 min read · Sep 8

4



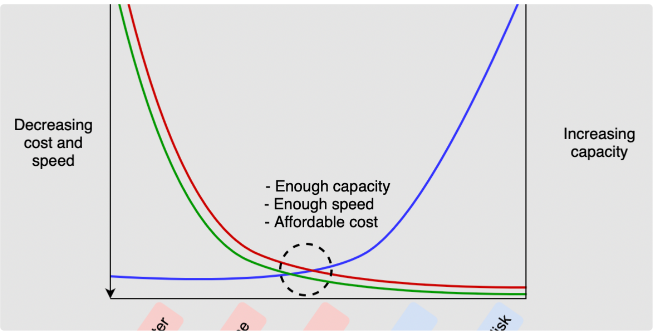
Lahiru_sujith

Distributed Systems Architectures..

What is Distributed System.?

7 min read · Sep 1

4



Suresh Podeti

Distributed Cache

Jonathan Fulton in Jonathan's Musings

AB Testing 101

Introduction

8 min read · Sep 22

7

What I wish I knew about AB testing when I started my career

17 min read · Aug 25

612 6

See more recommendations