INTERVIEW PACKET

BACKEND DEVELOPMENT – SDE 2

Interview Packet: Backend Development - SDE 2

Preface: Purpose of the document

The purpose of this document is to ensure anyone going for the interview at the company and position that this document is meant for, puts their best foot forward. This helps ensure only prepared learners are going for the interview, and the companies have less false negatives.

A smart and good fit learner failing the interview just because they could have prepared better is a loss for everyone.

What this document is for:

- Reducing false negatives in the interview process.
- Give a detailed walkthrough of the expected interview process.
- Give an exact idea of the difficulty level of the questions asked.
- Tips to help watch out for things to be aware of / careful about.

What this document is not:

- Repository of all actual questions asked in the interview.
- Trying to game the interview process.

This document is used to explain the interview process followed in **an app-based aggregator B2C company for Backend development roles**.

Evaluation Process

- Round 1: Online Assessment test. This is done on a platform like HackerRank/CodeSignal and is a 60-90 min test with 2 coding questions.
- Round 2: 60 minute DSA interview round. Conducted online. Medium level questions asked. Typically, 2 questions in the interview.
- Round 3: 60 minute DSA interview round. Conducted online/physical depending on the current guidelines in the company. Similar to Round 2, but focus on 1 medium hard question.
- Round 4: 60-75 mins LLD interview round. Focus on implementing code for a design pattern. Secondary focus on locks, mutex, concurrent programming.
- Round 5: 60-75 mins HLD interview round. Focus on designing a high scale system.
- Round 6: 60 min Hiring Manager round. Previous Project + Behavioral questions. Discuss previous work and projects. Deep dive into the decision-making process of the project, technologies used. What would have been done differently. Some behavioral questions.

Round-wise details

Round 1:

Mode:

Online. HackerRank-like online assessment.

Sample Questions:

```
Q1. Given an integer n representing a desired number of trips, and an array
cabTravelTime representing your cabs and how long it takes each cab (at that index of
the array) to make a trip, return the minimum time required to make n trips.
Assume that cabs can run simultaneously and there is no waiting period between trips.
There may be multiple cabs with the same time cost.*

Examples
If n=3 and cabTravelTime=[1,2], then the answer is 2. This is because the first cab
(index 0, cost 1) can make 2 trips costing a total of 2 time units, and the second
cab can make a single trip costing 2 at the same time.

n=10
cabTravelTime=[1,3,5,7,8]
* 7 trips with cab 0 (cost 1)
* 2 trips with cab 1 (cost 3)
* 1 trip with cab 2 (cost 5)
So, answer is 7 (there could be other combinations)

n=3
cabTravelTime=[3,4,8]
* 2 trips with cab 0 (cost 6)
* 1 trip with cab 1 (cost 4)
Time = 6

Q2. There is a long road with markers on it after each unit of distance. There are
some cars standing on the road. You are given the starting and ending coordinate of
each car (both inclusive).
Note: At any given marker there may be multiple cars or there may be none at all.

Your task is to find the number of markers on which at least one car is present. An
car with coordinates (l, r) is considered to be present on a marker m if and only if
l ≤ m ≤ r.

Example

For coordinates=[[4, 7], [-1, 5], [3, 6]], the output should be

easyCountcar(coordinates) = 9.
```

Sample answers:
For Q1, you can do a binary search for the answer. Use every cab for the number of trips possible in the current answer. If the total sum is more than desired n, current answer could be a possible answer. You binary search for the smallest possible current answer.

For Q2, this problem is exactly the same as merging intervals. Sort the ranges in increasing / decreasing order. Loop on them and keep merging the overlapping intervals. When the next one does not overlap, then add length of current merged interval to the answer and start afresh from the next entry (which did not overlap).

Why candidates fail:

- They start solving the hard problem first and end up spending all of the time on that problem.
- Not enough practice in a time constraint environment.
- Start solving the wrong problem statement. Take 3-5 mins to understand the problem statement. There is no rush.
- Start writing code before thinking of the best structure of the code. Take time to structure your code before starting to code.

Tips:

- Become comfortable with the coding editor of HackerRank and CodeSignal before the round.
- Practice in a time constraint environment.
- Address the issues mentioned in the section above.

Round 2:

Mode:
Online interview done on a shared coding panel / collabedit.

Sample Questions:
`Q1. Explain the solution to the problem 1 or 2 of the Round 1.`

```
Q2. Build a weighted random number generator. Assume you are given N numbers, and
with every number you also get the weight of that number. Higher the weight of a
number, higher should be the probability of that number being generated.

For example,
  Numbers: [1, 3, 5]
  Weights: [10, 1, 1]

Then you have to build a random number generator, where the output is either 1 (with
10/12 probability), or 3 (with 1/12 probability) or 5 (with 1/12 probability).

Code Expected
```

`Q3. https://leetcode.com/problems/pacific-atlantic-water-flow/ - Code not expected. Only the approach to solve.`

Sample answers:
Q1 is self-explanatory. Approached mentioned in earlier section.

Q2. You want to generate 1 number in the range of [1, Total sum of weights]. And then if range 1 - W[0] belongs to Num[0], W[0]+1 to W[0] + W[1] belongs to Num[1], … , you are basically searching for which range does the generated number belong to. You can use binary search / STL lower_bound for that.

Q3 solution: https://leetcode.com/problems/pacific-atlantic-water-flow/solutions/3266691/417-time-96-50-and-space-94-12-solution-with-step-by-step-explanation/

Why candidates fail:
- Use lower_bound but do not understand the internal implementation of lower_bound.
- Too slow to solve the first problem.
- Get stuck in the second problem.

Tips:
- Listen to hints being given.
- Name your functions properly.
- Structure your code well.
- Dry run your code on corner cases.

Round 3:
Mode:
Online/Offline interview with an interviewer 1:1. DSA.

Sample Questions:
```
Q1. Minimize the number of transactions to settle in Splitwise.
You are given amount owed between pair of friends.
For example, A -> B : 200, B -> C : 500, A -> B : 300
The above case can be handled by exactly one transfer of 500 from A->C.
Note that transfer is allowed between any pair of users.
```

Sample answers:
If you look at every single person, they need to pay some amount and receive some amount.
On an individual level, you can calculate the net amount (net amount is positive if I am owed money, negative if I owe money in total).
For those who have net amount as zero, no transactions are needed for them.
Everyone else can be broken down into 2 sets:
- Set 1 : People with +ve net money
- Set 2: People with -ve net money.

And then you settle in greedy way between the sets. More details at https://www.geeksforgeeks.org/minimize-cash-flow-among-given-set-friends-borrowed-money/

Why candidates fail:

- Miss out on corner cases
- Start thinking of a complicated graph algorithm.

Tips:

- Listen to hints being given.
- Name your functions properly.
- Structure your code well.
- Dry run your code on corner cases.

Round 4:

Mode:

Online/Offline interview with an interviewer 1:1. LLD.

Sample Questions:

```
Q1. Implement a multi-threaded pub-sub queue. Assume single machine.
Mention classes, functions, schema design of database, how you would handle
concurrent requests. SOLID principles, Design pattern to be used.
```

Sample answers:

Java Concurrency Interview Question: Multi-threaded Message Queue like Kafka, SQS, RabbitMQ
https://github.com/anomaly2104/low-level-design-messaging-queue-pub-sub

Why candidates fail:

- Don’t gather the requirements earlier. What are the features expected from the pub sub are not discussed with the interviewer, which leads to back and forth in later discussions. Often fail in figuring out following requirements:
    - There should be multiple topics.
    - Each subscriber should be sequential but multiple subscribers in parallel.
- Don’t use/ fail to mention the names of design patterns used. Observer pattern is a very important pattern used in coding PubSub.
- Aren’t aware of concurrency constructs required to code. Mostly need to be aware of concurrent data structures.

Tips:

- Start the interview by having a discussion on different requirements from the question. Jot down and clarify those requirements and do a handshake on those with the interviewer. Suggest different features wrt Message Queues to demonstrate your understanding about those and also to demonstrate how you may collaborate in a real work environment.

- Don't code in random order. Start by coding the classes of models (Publisher, Subscriber, Message) and then go to services and finally the main method that knits all together. This ensures the interviewer is in sync with you.
- Name variables, classes well. It shouldn't take much time to understand the purpose of a class/ variable.
- [Optional] Write test cases to demonstrate correct working and focus on best programming practices.

Round 5:

Mode:

Online/Offline interview with an interviewer 1:1. HLD

Sample Questions:

```
Q1. Design a system that implements a penalty/reward scheme for drivers. Rules for
penalty/reward are pre-specified. You get penalized per payout per ride if your
average rating drops below 4.0 out of 5.0. You get an incentive of 10% daily if you
complete 10+ rides on that day.

Deep dive into metrics to be measured for this system and the kind of alerts that
would be setup. How would those alerts be implemented? Pull/push?
```

Sample answers:

Rough outline:

Since the question only focuses on the penalty/reward system, we need not focus on other aspects (for example, rider to cab matching, cab status, path routing, etc.). Rating is easier, *so focus of this answer would only be on the daily incentive* (Do know that you would need to discuss rating related bonus as well).
Let's start with what is needed to be stored for calculating ratings and daily rides.
Question to the interviewer: At what frequency is the driver payment settled?
Assume the payments are settled weekly (only taking into account trips completed more than a day before the day of payment).

Relevant Schema:

- trips - id, driver_id, rider_id, cab_number, start_time, end_time, rider_rating, driver_rating, cab_fare

Since a trip can have multiple stops, best to have that in another table trip_stops. Similarly, if we want to support multiple riders (ride sharing), best to take that out in another table too and remove rider_id and rider_rating/driver_rating from trips table. Assumption that ride sharing is not supported.

- driver - id, name, photo-path, number-of-trips, rating
- driver_payout - trip_id, rating_penalty (default:0%), daily_bonus (default:100%), paid (default: false)

Daily bonus: A daily cron job which is scheduled to be executed once a day. Let's assume we post 1am everyday.

- Every cron job should be as lightweight as possible.

Cron job for all driver: In a single atomic query, you can find out drivers who have done more than 10 rides in the last day (where start_time > X and start_time < Y) and update driver_payout daily_bonus to 110% for those drivers. Something along the lines of

```
    UPDATE driver_payout dp
              JOIN trips t
                  ON t.id = dp.trip_id
                        AND t.start_time >= prev_day_start_time AND t.start_time <
prev_day_end_time
              SET  daily_bonus = 110
              WHERE  t.driver_id IN (
                                  SELECT driver_id, count(1) AS cnt
                                  FROM trips
                                  WHERE
                                        start_time >= prev_day_start_time
                                        AND start_time < prev_day_end_time
                                  GROUP BY driver_id
                                  HAVING cnt >=10
               )
```

Deep dive: What about the scale? Would this query not be very expensive?
Assumption: Storing all drivers and all rides in the same DB machine might not be feasible anyway. So, sharding is done based on the city. So, there will be a ***different cron job per city.*** A city would have 100k active drivers on any given day with an average of 1-10 rides per day. So, this query affects 1 million rows at most. 1 million row updates would make the above query very slow and would impact the performance of DB when this query is running.
Do we really need atomicity though? Since we are dealing with historical data (data more than an hour old and not changing), ***we don't need to keep things atomic***.

It is okay to run the select query and find the number of matching drivers (say 99000 drivers match). We can then process these drivers in chunks of 100 each.

- Would there be a problem if all users were processed in a single cron job? What if the machine fails? How do you recover?
- Since the operation is idempotent, update is to a fixed value of 110%, repeated updates are fine. However, there is a large cost to cron job failure because you would have to run the entire job for all 99k drivers all over again. Plus with so many drivers, chances are that you might run into memory issues. So, ***best to break it into separate cron jobs for X drivers (lets say 100 drivers each)***.
- Do you need to get the list of all 99000 drivers to schedule cron jobs for 100 each? Fetching all 99000 driver_ids might be expensive and can cause memory issues on the machine scheduling these cron jobs. If we sort by driver_id, and we assume there are pages of 100 sizes each, then can ***we schedule a job for each page***? We only need

the total number of matching drivers then (which is a single number). Each cron job gets their page number and hence can find their matching drivers based on page number. For example, drivers on page 3 can be fetched like the following:

```sql
SELECT driver_id, count(1) AS cnt
FROM trips
WHERE
      start_time >= prev_day_start_time
      AND start_time < prev_day_end_time
GROUP BY driver_id
HAVING cnt >=10
ORDER BY driver_id
LIMIT 100 OFFSET 200
```

Failure Handling:
All of this is great, but how do you effectively detect whether every cron job has succeeded. In fact, how do you even track that the master cron job which is supposed to schedule the other cron jobs is successful?
Firstly, how do you ensure cron jobs are reliably stored somewhere to be picked up by workers? Any ***messaging queue with persistence*** suffices (Kafka, RabbitMQ, Redis with persistence, etc.).
You have to assume the machine processing your cron job can fail in every way possible. So, you can't rely on the machine to re-queue the job. Then, how do you handle retries post failure?

Multiple ways to address this. One of the ways:
- A separate table with cron_job_id, arguments to cron, done/not_done/permanent_fail status.
    - Pre-filled with a master job for the current date.
    - The master job fills rows with entries of cron per page with page number in arguments.
    - An hourly / 3 hourly job which checks for entries that are "not done", and re-queues them in the messaging queue.

Issues to think about:
- Risk that if there is a job which takes a lot of time to run, it is still running and gets enqueued again because the job sees it's status as "not done". Highly unlikely in our case as no job should take more than a few seconds. Re-queuing is not disastrous as operations are idempotent.
- What if a code bug starts causing permanent failure in jobs. Re-queuing is anyhow an unexpected case and should send a notification to the service owner.
    - There should be an alert sent for permanent failure status in the table.
    - Should alerts be push / pull - If they are pull, then you are relying on the owners to be proactively checking status. Best if it is push, so issues are immediately escalated.

Why candidates fail:

- They go too wide - trying to design an entire product instead of focusing on what the interviewer wants to focus on.
- They only answer the question asked and then expect the interviewer to poke further.
- Not considering failure cases. Assume only the happy case. What if the cron jobs fail? How do you reliably retry?
    - Assume a master cron job will always be successful.
- Large heavy jobs - While they may be fine for some cases, it is important to validate with scale that the cron jobs you run will be lightweight and have low chances of failing.
- Only define how the alerts will be defined. Don't elaborate on pushing critical alerts (set subscription for alerts).

Tips:

- Drive the discussion. Actively state what you are thinking - pros and cons of the approach you are suggesting.
    - This does not mean that you should continuously be talking. Think and then speak. But drive the discussion. Interviewer shouldn't have to keep poking questions to keep the discussion moving.
- Actively listen to the hints that the interviewer might be giving.
- Clarify the requirements of the question -> Quickly estimate the scale for which you are building -> Understand what design goals and then jump into the actual design. First 3 sections should not take more than 5-10 mins.

Round 6:

Mode:

Online/Offline interview with an interviewer 1:1. HLD

Sample Questions:

[Behavioral]

Q1. Tell me about a time when you had very insightful comments on a code review. What did you learn in the process?

When was the last time you left an insightful comment / set of changes on a code review?

Q2. Tell me about a time when you disagreed with your manager. How was that resolved?

[Past Projects]

Most challenging project.

If you were redesigning again, how would you redesign the entire project? Why?

Sample answers:

Answers are subjective to a person's profile.

Why candidates fail:

- Claim that there was never a disagreement with the manager. Shows you don't have an opinion. It's good to have an opinion, however not a rigid one. Everyone in the team should be able to reason out on the best approach.
- You say that the project would look exactly the same even if you were redesigning today. Shows you have not learnt during the project. No project is perfect.
- You mention fancy technology names on your resume, but do not understand how they work internally.
- You don't show passion for the work you have done - this reflects if you don't have much to talk about your previous work, or you talk about it with disinterest.

Tips:

- Think about the above questions in detail beforehand. What would you change if you were doing projects you have done in the past again.
- Research about all technologies you have mentioned on your resume. If you mention Kafka, you will be questioned on the internals of Kafka, or how Kafka can be used in a certain scenario.
- Think about healthy disagreements with peers, managers and reportees (if applicable). Exact arguments for and against the point made. How did you arrive at a consensus?
- Actively think about how you have grown in the last 1 year, last 2 years, last 6 months. What are the new things you have learnt - both skills wise, and behavior wise.