Design prompts

- submitted FRs and NFRs in a pdf doc to claude
- added 2 minute reservation hold functionality into the FRs
- Asked claude to make a high level architecture to implement these requirements and give reasons as to why alternatives were rejected
- It created the first file (SYSTEM_ARCHITECTURE.md)
- asked it to generate a more detailed document with deeper analysis and more mathematical crunching to support mentioned requirements.
- It created the second file (SYSTEM_ARCHITECTURE_ULTRA.md)
- Changed implementation scope to single region (Assumption - sale and shipping will be help in a single region, although this can be scaled to a multi region architecture with each regional cell hosting and handling its own flash sale with isolated inventory) ”one more edit, the system is supposed to host the sale in a single region only (APAC). Please make required changes to the entire doc accommodating this change.”
- Asked to add product search with Elasticsearch + Redis cache
- **Asked to implement a better, Three-layer reservation expiry system(2 minute hold)**
- Corrected Claude’s understanding of partitioning 250K RPS into reads and writes whereas our specifications were different.

  "To handle the requirement of supporting 250K reads, we have currently assumed 95% reads and 5% writes. What is this assumption based on ? We have separate requirements for reads as 250K RPS and writes at 25K RPS (reservations)"
- Challenged and corrected the split between availability of stock and product info Requests. It assumed for 80% reads to be product info (which is static cacheable content).

  Prompt - "To address the requirement of supporting 250K reads, we have assumed 80% product details reads and 20% availability reads. What is this assumption based on ?"

  The impact of stale cache was lower in this assumption and the proposed design involved reading availability from replicas, so once the assumption was changes, the number of reads were much higher, it became apparent that even s small percentage of users seeing stale cache will affect thousands of users. hence the approach was pivoted to use master Redis node to read.

  Prompt - "after we pivoted our approach to do reads and writes from master, instead of read replicas, we haven't updated the previous mentions of read-from-replica enabled. For instance, in the selected line. Let's remove all references of reading from replica for the caching solution to support 200K RPS reads of availability"

Before the sale starts, most people have already used cache and loaded product information, the traffic at the time of sale will be mostly about availability check (write).

This then lead to a significant change in the architecture, and involved having to vertically scale up Redis instance to support higher real time availability read throughput.

- Upon deliberating what the batch size should be, the decision depeneded upon tradeoffs between size of the batch vs wait time to fill up each batch. Given a realistic scenario where at the same moment only about 1200-1500 requests qould line up, a batch of 250 seemed to handle the traffic without causing more than 50ms latency in worst cases.
- In the design doc claude had claimed max waiting time for a user request in the reservation kafka topic would be around 50ms. I asked the rationale behind this and it proposed a realistic scenario where reservation requests would come in bursts of 1200-1500 at any given instance, which could be handled with a batch size of 250 and the maximum time a request would have to wait in the queue would only add 50ms more latency because each batch processing time is 10ms.
- Found issues where in the caching strategy, in the redis cluster, read from replicas for reads requests was generating stale cache in those reads, changed all reads and writes to be done from a single redis node while replicas were to be used for just analytics purpose and to maintain High Availability
- cache layer achitecture was assuming 250k RPS including reads and writes. we then asked it to design for handling 250k RPS reads from cache redis and 25k writes by single writer pattern in kafka topics partitioned by SKU_ID
- Implemented rate limiting for read APIs as well The design proposed by claude had built a sophisticated architecture to throttle write APIs but overlooked rate limiting to also be executed on read APIs to ensure system durability

  Prompt - "i think we should also look at throttling or rate limiting the read requests to avoid DDoS attacks and other similar attacks. since many people will try to open product pages for the flash sale, we can allow much higher token limits and thresholds for read requests."
- This design handles upto 100 different SKU_IDs each of which can handle the peak load of 250k RPS read and 25K RPS write requests. (The kafka topics have been partitioned per SKU therefore the FIFO order will be maintained for fairness). We can scale back to 1 product support as well if that is our requirement
- All these further clarifications and modifications to the doc resulted in a more comprehensive and fundamentally sound design document which we updated to **SYSTEM_ARCHITECTURE_ULTRA_V2.md**
- After thoroughly checking the flow of architecture, giving prompts to improve the accuracy on scale and performance/throughput bottleneck handling, created a finalized **FLASH_SALE_SOLUTION_DESIGN.md** file to concentrate all selected decisions and plans into a single consolidated document.

  Prompt given -

This document was created with the purpose of formulating the plan to host a flash sale event, with some functional and non functional requirements, and coming up with a viable system architecture to implement it after comparing various possible alternatives and approaches to various ddecision layers.

Now that we have formulated the approach pretty much, lets write down our solution in a much better flow, starting with

1. stating the problem statement
2. requirements
3. A high level overview of the proposal,
4. A high level system architecture diagram,
5. A list of proposals stating every problem and its solution approach, with alternatives listed in the appendix down somewhere.