Customer Obsession

Leaders start with the customer and work backwards. They work vigorously to earn and keep customer trust. Although leaders pay attention to competitors, they obsess over customers.

Story 1 - Balancing Customer Needs with Business Goals

When designing the grading workflow, a conflict arose between **real-time grading feedback for students** (customer need) and the cost of using LLM APIs for every submission (business goal). Real-time feedback was critical for customer satisfaction, but the high API cost posed a challenge for scalability. To strike a balance, you proposed a hybrid solution where objective answers (like multiple-choice) were graded locally using in-house logic, while subjective answers were processed via the Gemini API. Additionally, you used **Kafka retry mechanisms** to stagger API calls, which reduced costs without compromising on the feedback quality. The solution satisfied both customer expectations and business constraints.

Story 2 - Incorporating Customer Feedback into Product Improvements

While testing the **Student Management module**, early feedback from administrators and teachers highlighted the need for **soft delete** functionality (to retain data for audits) rather than the initially planned hard delete. Although the original design prioritized simplicity, you recognized the importance of aligning with customer needs. You revisited the implementation, enabling soft delete with minimal disruption to the development timeline. The change was well-received by users, who appreciated the flexibility, and the adjustment became a key selling point for the platform.

Can you tell me about a time when you went above and beyond to satisfy a customer?

Situation:
During the implementation of the **Examination Service**, I encountered an issue with the consistency of AI grading using the Gemini API. The system was designed to provide feedback on subjective answers, but during testing, I noticed that similar student answers were getting varying scores. This inconsistency risked frustrating teachers and students, who expected fair and reliable grading.

Task:
My goal was to ensure that the grading system delivered accurate and consistent results to build trust with both teachers and students, even if it required extra effort.

Action:
I dedicated additional time to analyze discrepancies by running multiple test cases and identifying patterns in how the API was grading answers. I refined the prompts sent to the Gemini API to make them more structured and precise. To further address variability, I developed a **normalization logic** that adjusted scores to reduce outliers and ensure fairness. Additionally, I communicated transparently with stakeholders, updating them on progress and showing them examples of improvements as I iterated.

Result:
The grading system became significantly more consistent, and teachers appreciated the reliability of the feedback. This improvement boosted their confidence in the platform, and students also felt reassured by the fairness of the grading. Stakeholders specifically highlighted this enhancement as a major value-add for the service.

Describe a situation where you had to balance customer needs with business goals. How did you handle it?

Situation:
In the **Examination Service**, there was a conflict between providing **real-time grading feedback** to students (a critical customer need) and managing the high costs associated with using the Gemini API for every submission (a key business goal). Real-time feedback was important for students' learning experience, but the API costs during peak submission periods posed scalability challenges.

Task:
I needed to find a solution that met students' expectations for timely feedback while keeping operational costs manageable for the business.

Action:
I proposed a hybrid approach: For objective answers (like multiple-choice), I implemented local grading logic to process them in-house instantly. For subjective answers, I used the Gemini API but optimized the workflow to minimize costs. Specifically, I integrated **Kafka's retry mechanism** to stagger API requests, preventing overload during peak times, and implemented batching where possible. This allowed the system to handle large-scale submissions efficiently without sacrificing the quality of feedback.

Result:
The solution provided a balanced outcome—students received real-time feedback for objective questions and timely, detailed feedback for subjective answers, while the API costs were significantly reduced. Both students and stakeholders were satisfied, and the approach scaled effectively during high-volume exam periods.

How do you ensure that customer feedback is incorporated into product development or improvements?

STAR Answer:

Situation:
While developing the **Student Management module**, administrators and teachers requested a **soft delete** feature to retain deleted student records for audits or recovery, instead of the initially planned **hard delete**, which would permanently erase records. The hard delete design was simpler and aligned with our original timeline, but the feedback made it clear that soft delete would better meet their needs.

Task:
I needed to incorporate this feedback into the system without causing significant delays or disrupting the project timeline.

Action:
I reassessed the system design and modified the data model to support **soft delete**, where records are flagged as deleted but not permanently removed. I updated the API logic and adjusted the filtering criteria for read operations to ensure that only active records were shown to users by default. Throughout the process, I maintained open communication with stakeholders, sharing how the feature was being implemented and gathering additional input to ensure it met their requirements.

Result:
The soft delete feature was successfully implemented with minimal disruption to the timeline. Administrators and teachers were highly satisfied, as the feature gave them the flexibility to recover deleted records when needed. This not only met their needs but also became a key selling point for the platform, showcasing our responsiveness to customer feedback.

Ownership

Leaders are owners. They think long-term and do not sacrifice long-term value for short-term results. They act on behalf of the entire company, beyond just their own team. They never say, "that's not my job."

Story 1: Ownership of Request Tracing for Logging Consistency (Centralized Logging)

While working on the centralized logging system for the HRMS microservices, you noticed that debugging was becoming a bottleneck due to inconsistent logging formats and the inability to trace requests across services. Although this was not explicitly assigned to you, you took ownership of standardizing the logging format across all microservices and implementing request tracing using unique request IDs. You also centralized the logs into Kibana, making them searchable across the system. The result was a 50% reduction in debugging time and faster incident response, which significantly improved the team's productivity and system reliability.

Story 2: Taking Ownership of Grading Workflow Scalability (Online Examination Evaluator)

During the development of the **Online Examination Evaluator**, you realized that the grading workflow using the Gemini API could face scalability issues during high traffic periods, especially during exam submissions. This was not something your team had planned for initially, but you proactively took the lead in designing a scalable solution. You integrated Kafka as a messaging system to queue and process submissions asynchronously, implemented batching to optimize API usage, and fine-tuned the retry mechanism to avoid overloading the system. This ensured that the grading workflow could handle peak loads without service interruptions, and the solution was implemented without impacting project timelines.

Tell me about a time you took ownership of a project or problem. What was the outcome?

Situation:
While working on the company's HRMS system, we faced a major bottleneck in debugging issues across multiple microservices like employee profiles, payroll, and attendance. Each service had inconsistent logging formats, lacked request tracing, and provided no way to trace errors across the system. This caused significant delays, particularly during payroll runs, which were time-sensitive and critical for employee satisfaction.

Task:
Although I wasn't explicitly assigned to fix this, I knew the issue was affecting the team's productivity and the reliability of the system. I decided to take ownership of the problem and deliver a standardized, scalable logging solution that would allow us to trace requests across services and debug issues more efficiently.

Action:
I designed a standardized logging format that included essential details like request IDs, timestamps, service names, and error messages. I implemented a mechanism to generate and propagate unique request IDs across all microservices, enabling end-to-end tracing. Then, I centralized the logs into Elasticsearch and created a user-friendly dashboard in Kibana to make them easily searchable.
To ensure minimal disruption to ongoing work, I collaborated with other developers to incrementally roll out the changes, created adapters for older services, and provided detailed documentation and training.

Result:
The solution reduced debugging time by nearly **50%**, as developers could now trace issues end-to-end across services in a few clicks. Incident response times improved significantly, and the system became much more reliable. HR and Ops teams were especially satisfied, as payroll-related errors could now be resolved quickly, ensuring timely payments to employees. The project was recognized internally as a key improvement to the HRMS platform.

Question 2: Can you describe a situation where you identified an issue or improvement opportunity and took it upon yourself to address it?

Situation:
During the development of the **Online Examination Evaluator**, the system relied on Google's Gemini API to grade subjective answers. As the system neared deployment, I realized that during peak exam periods, the API usage could exceed rate limits or cause delays, creating a bottleneck in grading workflows. This issue had not been identified during the initial design phase but could severely impact user experience and system reliability.

Task:
I took it upon myself to design a scalable and reliable grading workflow that could handle peak loads without interruptions, even though it wasn't part of my assigned tasks.

Action:
I integrated **Kafka** as a messaging system to queue and process exam submissions asynchronously, ensuring that requests were handled efficiently without exceeding API rate limits. I implemented **batching** of requests to reduce the number of API calls and optimized the **retry mechanism** to handle failures gracefully. To minimize latency, I fine-tuned the consumer logic and adjusted Kafka configurations to support high-throughput environments. I also worked closely with the DevOps team to monitor performance during load tests and make necessary adjustments.

Result:
The grading workflow became highly scalable and could handle peak loads without downtime. By proactively addressing this issue, I ensured the system met both technical and customer expectations. This improvement allowed us to deploy the system on time and

deliver a seamless grading experience, even during high-traffic periods. The solution was later adopted as a best practice for other high-load workflows in the organization.

Question 3: How do you handle situations where you don't have direct authority but still need to take ownership of an outcome?

(Story: Ownership of Request Tracing for Logging Consistency – Adapted)

STAR Answer

Situation:
Our HRMS platform had critical issues with debugging due to inconsistent logging formats across its microservices. This resulted in significant delays during payroll runs and compromised the system's reliability. I realized that standardizing and centralizing logs would solve this problem. However, since I didn't have direct authority over the other microservice teams, I needed to influence them to adopt the changes without disrupting their workflows.

Task:
My goal was to take ownership of the issue by standardizing logging across services and implementing request tracing, while working collaboratively with other teams to ensure their buy-in and participation.

Action:
I began by designing a simple and scalable logging format that included critical information like request IDs and timestamps. To gain support, I organized a series of meetings to explain the benefits of this solution and how it would help the entire team debug faster and improve system reliability. I also created a detailed implementation plan, showing how the changes could be rolled out incrementally with minimal disruption. For older services using legacy frameworks, I developed adapters to make adoption easier. Throughout the process, I kept an open line of communication with the teams, sought feedback, and provided documentation and training to help them transition smoothly.

Result:
The standardized logging format was successfully implemented across all services, resulting in a **50% reduction in debugging time** and faster incident resolution. Even though I didn't have formal authority, my ability to clearly communicate the value of the solution and provide practical implementation support won the trust and cooperation of the other teams. The project became a cornerstone improvement for the HRMS system and was well-received across the organization.

Invent & Simplify

Leaders expect and require innovation and invention from their teams and always find ways to simplify. They are externally aware, look for new ideas from everywhere, and are not limited by "not invented here." As we do new things, we accept that we may be misunderstood for long periods of time.

Story 1: Using Kafka for Scalable, Asynchronous Grading (Exam Evaluator)

You foresaw performance bottlenecks with real-time grading of exams via the Gemini API. Instead of building a heavyweight solution, you implemented a **Kafka-based architecture** to decouple grading from submission. This made the system **asynchronous, fault-tolerant**, and easier to maintain, while keeping latency manageable and code modular.

Story 2: Indexing + Redis Caching to Speed Up Reporting (Analytics Service)

The system was struggling to generate real-time reports due to millions of student activity logs. You introduced **timestamp indexing** and **Redis caching**—a lightweight, non-invasive solution that didn't require database redesign but delivered massive speed gains. This empowered teachers to take timely actions.

Story 3: Hybrid Evaluation Strategy to Reduce LLM Costs (Exam Evaluator)

Grading subjective answers with LLMs (Gemini API) was **expensive at scale**. You proposed a **hybrid grading model**—automating objective question evaluation with local logic and using the LLM only for subjective answers. You also added **batching and retry strategies**. This innovation cut costs significantly without compromising on grading quality or UX.

Describe a time when you had to simplify a complex process or system. What did you do, and what was the result?

Situation:
In our AI-driven online examination system, students submitted subjective answers that were graded using Google's Gemini API. However, processing each submission in real time created serious risks of **rate-limiting**, latency, and potential downtime—especially during peak exam windows.

Task:
I needed to design a **simple but scalable** system to decouple submission from grading, while preserving grading accuracy and user experience.

Action:
I re-architected the grading workflow using **Kafka**. Instead of processing submissions immediately, I pushed each submission as an event to a **Kafka topic**. I developed a **consumer service** that asynchronously pulled these events and made API calls to the LLM, adding retry logic to handle failures. This simplified the submission endpoint to just pushing an event—reducing coupling and latency—and made the grading process resilient, testable, and modular.

Result:
The system handled exam submissions from thousands of students **without delay or degradation**, even under peak load. The solution was simple to maintain and significantly improved reliability without overcomplicating the backend. It became a foundational pattern reused in future services.

Tell me about an innovation or change you implemented that made things easier for your team or company.

Situation:
A large school using our examination platform was facing long delays in generating student performance reports due to millions of log records generated monthly. These delays made it difficult for teachers to identify students who needed intervention.

Task:
I was responsible for improving report generation speed **without overhauling the database schema or backend architecture**.

Action:
I observed that the reporting system heavily relied on **range queries by timestamp**, so I added indexes to frequently queried fields like timestamps and student IDs. Then, I implemented **Redis caching** for high-frequency reports like weekly attendance and submission summaries. This drastically reduced the database load and allowed reports to load instantly if cached. I also configured cache expiry policies to keep data accurate.

Result:
Uncached reports became 75% faster (down to ~1.5 seconds), while cached reports loaded in **<100ms**. This allowed teachers to act on insights in real time and significantly improved the system's usability. The change was simple to implement and highly impactful.

Can you share an example where you came up with a creative solution to a difficult problem?

Situation:
Our examination system used an LLM (Gemini API) to grade student answers. While effective, it became clear that **grading all responses through the API would be costly and possibly unsustainable at scale**, especially as the user base grew.

Task:
I was responsible for ensuring that we could scale the grading process without compromising quality—or budget.

Action:
I proposed a **hybrid evaluation strategy**:

- Objective questions were graded locally using simple rules and logic.
- Subjective answers were routed through the Gemini API.
- I also implemented **batching of API requests** and used Kafka's retry system to avoid redundant API calls during spikes.

This reduced the volume of LLM usage by over 60%, without any impact on accuracy or UX. It allowed us to maintain fast turnaround times and handle high exam volumes on a leaner infrastructure.

Result:
We **reduced API costs significantly**, maintained grading quality, and improved throughput during peak loads. This creative, hybrid approach became a best practice internally and allowed us to grow without budget constraints.

Are Right, A Lot

Leaders are right a lot. They have strong judgment and good instincts. They seek diverse perspectives and work to disconfirm their beliefs.

Story 1: Using Kafka for Scalable, Asynchronous Grading (Exam Evaluator)

You foresaw performance bottlenecks with real-time grading of exams via the Gemini API. Instead of building a heavyweight solution, you implemented a **Kafka-based architecture** to decouple grading from submission. This made the system **asynchronous, fault-tolerant**, and easier to maintain, while keeping latency manageable and code modular.

Story 2: Indexing + Redis Caching to Speed Up Reporting (Analytics Service)

The system was struggling to generate real-time reports due to millions of student activity logs. You introduced **timestamp indexing** and **Redis caching**—a lightweight, non-invasive solution that didn't require database redesign but delivered massive speed gains. This empowered teachers to take timely actions.

Story 3: Hybrid Evaluation Strategy to Reduce LLM Costs (Exam Evaluator)

Grading subjective answers with LLMs (Gemini API) was **expensive at scale**. You proposed a **hybrid grading model**—automating objective question evaluation with local logic and using the LLM only for subjective answers. You also added **batching and retry strategies**. This innovation cut costs significantly without compromising on grading quality or UX.

Describe a time when you had to simplify a complex process or system. What did you do, and what was the result?

Situation:
In our AI-driven online examination system, students submitted subjective answers that were graded using Google's Gemini API. However, processing each submission in real time created serious risks of **rate-limiting**, latency, and potential downtime—especially during peak exam windows.

Task:
I needed to design a **simple but scalable** system to decouple submission from grading, while preserving grading accuracy and user experience.

Action:
I re-architected the grading workflow using **Kafka**. Instead of processing submissions immediately, I pushed each submission as an event to a **Kafka topic**. I developed a **consumer service** that asynchronously pulled these events and made API calls to the LLM, adding retry logic to handle failures. This simplified the submission endpoint to just pushing an event—reducing coupling and latency—and made the grading process resilient, testable, and modular.

Result:
The system handled exam submissions from thousands of students **without delay or degradation**, even under peak load. The solution was simple to maintain and significantly improved reliability without overcomplicating the backend. It became a foundational pattern reused in future services.

Tell me about an innovation or change you implemented that made things easier for your team or company.

Situation:
A large school using our examination platform was facing long delays in generating student performance reports due to millions of log records generated monthly. These delays made it difficult for teachers to identify students who needed intervention.

Task:
I was responsible for improving report generation speed **without overhauling the database schema or backend architecture**.

Action:
I observed that the reporting system heavily relied on **range queries by timestamp**, so I added indexes to frequently queried fields like timestamps and student IDs. Then, I implemented **Redis caching** for high-frequency reports like weekly attendance and submission summaries. This drastically reduced the database load and allowed reports to load instantly if cached. I also configured cache expiry policies to keep data accurate.

Result:
Uncached reports became 75% faster (down to ~1.5 seconds), while cached reports loaded in **<100ms**. This allowed teachers to act on insights in real time and significantly improved the system's usability. The change was simple to implement and highly impactful.

Can you share an example where you came up with a creative solution to a difficult problem?

Situation:
Our examination system used an LLM (Gemini API) to grade student answers. While effective, it became clear that **grading all responses through the API would be costly and possibly unsustainable at scale**, especially as the user base grew.

Task:
I was responsible for ensuring that we could scale the grading process without compromising quality—or budget.

Action:
I proposed a **hybrid evaluation strategy**:

- Objective questions were graded locally using simple rules and logic.
- Subjective answers were routed through the Gemini API.

- I also implemented **batching of API requests** and used Kafka's retry system to avoid redundant API calls during spikes.

This reduced the volume of LLM usage by over 60%, without any impact on accuracy or UX. It allowed us to maintain fast turnaround times and handle high exam volumes on a leaner infrastructure.

Result:
We **reduced API costs significantly**, maintained grading quality, and improved throughput during peak loads. This creative, hybrid approach became a best practice internally and allowed us to grow without budget constraints.

Learn and be curious

Leaders are never done learning and always seek to improve themselves. They are curious about new possibilities and act to explore them.

Story 1: Learning Kafka + Event-Driven Architecture for Exam Grading

When faced with the challenge of scaling the LLM-based exam grading system, you realized the synchronous grading approach wouldn't work under load. To solve this, you chose to use **Kafka for asynchronous processing**—a technology you hadn't worked with extensively before. You took the initiative to learn Kafka fundamentals, producer/consumer patterns, and message reliability strategies. As a result, you successfully implemented a decoupled, scalable grading pipeline that performed reliably under high traffic.

Story 2: Asking Deep Questions About Report Delays (Analytics Service)

When the platform was experiencing long delays in generating reports, most assumed the system just needed more hardware or database upgrades. Instead of jumping to a solution, you **dug deeper by analyzing query patterns**, asked questions about how teachers were using the reports, and uncovered that the problem stemmed from **timestamp-based full table scans**. This led you to implement indexing and caching—a solution that was both simple and highly effective.

Question 1: Can you share an example of a situation where you had to learn something new to be successful in your role?

Story: Learning Kafka + Event-Driven Architecture for Exam Grading

STAR Answer:

Situation:
While working on the **Online Examination Evaluator**, we needed to grade subjective answers using the Gemini LLM API. Initially, the grading logic was synchronous—API calls were made immediately when students submitted their answers. But as we simulated higher user loads, I realized this approach would break under pressure due to **rate limits, latency, and lack of resilience**.

Task:
My task was to **design a scalable and fault-tolerant grading system**. Although Kafka was proposed as a solution, I hadn't used Kafka in a production setting before. I knew I had to quickly ramp up to make the right design decisions and implement them correctly.

Action:
I took the initiative to **learn Kafka's architecture**, focusing on producer-consumer patterns, message durability, and error handling. I consumed Kafka documentation, tutorials, and GitHub samples over a few days and validated concepts by building prototypes.
Once confident, I re-architected the grading workflow: exam submissions were published to a Kafka topic, and a consumer service handled grading asynchronously—complete with **retry logic and batching**. I also fine-tuned message throughput and error handling for production readiness.

Result:
The final implementation was **scalable, reliable, and cost-efficient**. It smoothly handled thousands of exam submissions, even during peak times, and became a reusable pattern for other async processes in our system. The project also boosted my long-term confidence in distributed systems design, and Kafka is now a core part of my engineering toolkit.

Question 2: Tell me about a time when you asked questions to better understand an issue and drove a better solution.

Story: Asking Deep Questions About Report Delays (Analytics Service)

STAR Answer:

Situation:
In the analytics service of our e-learning platform, teachers and administrators were experiencing **long delays when generating reports**—some taking up to 8 seconds. There was growing pressure to improve this, and the initial suggestion from some team members was to increase server capacity or migrate the database.

Task:
Before committing to a costly infrastructure upgrade, I took ownership of understanding the root cause. My goal was to **ask the right questions** to figure out why report generation was slow and whether there were more effective solutions.

Action:
I started by analyzing **query logs and report usage patterns**. I asked:

- What fields were most commonly filtered in the reports?
- Were these full-table scans or index-optimized queries?
- How often were reports being requested?
  These questions led me to discover that the majority of queries involved **range-based filtering on timestamps**, and no indexes were present on those fields. I also noticed certain reports were being fetched repeatedly with the same parameters.

Based on that, I implemented **indexing on timestamps and student IDs**, and added a **Redis caching layer** for frequently requested reports. I configured cache expiry policies to ensure accuracy without overloading the DB.

Result:
Report generation speed improved dramatically:

- **75% faster** for uncached reports (1.5–2 seconds).
- **>95% faster** for cached reports (<100ms).
  The changes not only improved teacher experience but also avoided a costly and unnecessary infrastructure upgrade. This experience reaffirmed for me the power of asking deep questions and validating assumptions before acting.