Recitation Notes:

GQM Activity

1. **Performance**

Goal: Ensure the system processes a high volume of payment transactions quickly and reliably.

Question	**Metric**
Q1: How many transactions can the system handle per second without exceeding acceptable response times?	- **TPS** under peak load - Average / 95th percentile / 99th percentile response time (ms)
Q2: Does the system degrade gracefully under heavy load (spikes, seasonal peaks)?	- **Error rate (%)** during peak load - Queue length / backlog size if using asynchronous queues
Q3: How does latency vary across different geographies?	- **Response time by region** (e.g., North America, Europe, Asia) - Latency difference between regions

2. **Availability**

Goal: Maximize service uptime so that merchants and customers can process payments anytime.

Question	**Metric**
Q1: How often is the system accessible to end users and merchants?	- **Uptime (%)** over a defined period (e.g., monthly/quarterly) - Number of downtime incidents
Q2: How quickly does the system recover from unexpected failures (hardware, software)?	- **Mean Time to Restore (MTTR)** - Mean Time Between Failures (MTBF)

Question	Metric
Q3: Do planned maintenance windows disrupt normal payment traffic?	- **Duration of scheduled maintenance** - Number of transactions affected

---

3. **Security**

Goal: Protect the payment platform and user data against unauthorized access, fraud, and data breaches.

Question	Metric
Q1: How many security vulnerabilities or breach attempts are detected and mitigated?	- **Number of detected intrusion attempts per month** - Number of reported security vulnerabilities (internal or external)
Q2: How frequently and quickly are known vulnerabilities patched?	- **Time to patch/remediate** (days/hours) - Number of unpatched critical vulnerabilities
Q3: Is cardholder data or personally identifiable information (PII) secured adequately?	- **Compliance checks** (PCI DSS, GDPR, etc.) - Encryption coverage (e.g., % of data encrypted at rest/in transit)
Q4: How effective is the fraud detection mechanism?	- **False positive rate** (legitimate transactions flagged) - Chargeback ratio (disputed transactions vs. total transactions)

---

4. **Scalability**

Goal: Allow the system to handle growth in number of users, transactions, and integration points without significant performance loss or prohibitive cost increases.

Question	**Metric**
Q1: How does throughput (TPS) scale with additional compute resources (e.g., more servers, containers)?	- **Horizontal scalability ratio** (TPS increase vs. server count) - Resource utilization (CPU, memory) under varying loads
Q2: How does cost grow relative to transaction volume?	- **Cost per transaction** (infrastructure + operational costs) - Cost elasticity (ΔCost ÷ ΔLoad)
Q3: Can new regions (data centers) be added to reduce latency?	- **Time to provision** additional regions - Latency reduction observed after spinning up new region

5. **Maintainability**

Goal: Ensure the system can be easily updated, extended, and debugged with minimal disruption.

Question	**Metric**
Q1: How long does it take to identify and fix bugs or issues in production?	- **Mean Time to Detect (MTTD)** - Mean Time to Resolve (MTTR) for defects
Q2: How quickly can new features or payment methods be rolled out?	- **Deployment frequency** - Lead time for changes (from code commit to production)
Q3: How modular is the codebase to support partial updates?	- **Cyclomatic complexity** or other code metrics

Question	Metric
	- Number of modules with lines of code or a single monolith size
Q4: How effective is the testing strategy to prevent regressions?	- **Automated test coverage (%)** - Number of critical defects found post-deployment

---

6. **Reliability**

Goal: Ensure the system consistently processes valid transactions and resists data corruption or inconsistent states.

Question	Metric
Q1: How often do payment transactions fail due to internal errors?	- **Transaction success rate** (%) - Internal error rate (# errors / total transactions)
Q2: Do partial failures cause incorrect balances or lost transaction data?	- **Number of data inconsistency incidents** - Recovery time for data reconciliation after partial failures
Q3: Is the system resilient to hardware or network outages?	- **Fault tolerance tests** (e.g., chaos engineering) pass/fail rate - RPO/RTO for critical transaction data

QAS Activity

1. Performance

Scenario P1

- **Source**: A large number of customers attempting to check out simultaneously
- **Stimulus**: 10,000 transactions are initiated within a 1-minute window (peak holiday surge)
- **Artifact**: Payment Processing Service, Database
- **Environment**: Production environment, standard operations, external payment gateways active
- **Response**: The system processes each transaction request and responds without timing out
- **Response Measure**:
    - **Average latency** under 2 seconds per request
    - **Error rate** < 1% during the peak load

Scenario P2

- **Source**: Automated load testing tool
- **Stimulus**: Sustained throughput of X transactions/second over 30 minutes
- **Artifact**: Entire Payment Portal stack (web tier, application tier, DB tier)
- **Environment**: Staging environment configured similarly to production
- **Response**: System handles sustained load without performance degradation
- **Response Measure**:
    - **95th percentile response time** < 3 seconds
    - **No critical performance alerts** (CPU < 80%, memory < 75%)

---

2. Availability

Scenario A1

- **Source**: Network failure in one data center
- **Stimulus**: A major ISP outage causes the primary data center to lose connectivity
- **Artifact**: Payment Processing Service, Merchant Portal
- **Environment**: Production environment, peak business hours
- **Response**: The system automatically fails over to a secondary data center

- **Response Measure**:
    - **Recovery Time Objective (RTO)** ≤ 2 minutes
    - **Number of lost or stalled transactions** < 0.1%

Scenario A2

- **Source**: Infrastructure maintenance
- **Stimulus**: Rolling server updates or patches are applied
- **Artifact**: Merchant onboarding and user authentication services
- **Environment**: Off-peak hours in production
- **Response**: Zero downtime deployment ensures system remains accessible
- **Response Measure**:
    - **Uptime** ≥ 99.9% during maintenance window
    - **No user login failures** or broken sessions

---

3. Security

Scenario S1

- **Source**: Malicious actor or botnet
- **Stimulus**: High-volume fraudulent transactions or brute-force attempts on login endpoints
- **Artifact**: Authentication component, Fraud Detection service
- **Environment**: Production environment under moderate load
- **Response**: System detects unusual patterns, blocks suspicious IPs or accounts, and triggers alerts
- **Response Measure**:
    - **Percentage of fraud attempts blocked** ≥ 95%
    - **False-positive rate** < 5%
    - **Security alerts** raised to on-call team within 1 minute of detection

Scenario S2

- **Source**: Quarterly PCI DSS compliance audit
- **Stimulus**: Auditor requests evidence of data encryption and security posture
- **Artifact**: Stored cardholder data, transaction logs
- **Environment**: Normal production environment
- **Response**: The system demonstrates compliance via encryption at rest and in transit, secure access controls
- **Response Measure**:
    - **Successful PCI DSS certification**
    - **Zero critical findings** in the audit report

---

4. Scalability

Scenario SC1

- **Source**: Marketing campaign causing a sudden influx of new merchants
- **Stimulus**: 500 new merchants sign up each minute and start processing transactions
- **Artifact**: Merchant Onboarding Service, Payment Processing, DB clusters
- **Environment**: Production environment, standard usage patterns plus sudden spike
- **Response**: Platform scales horizontally (more app server instances, DB shards) to handle increased load without performance loss
- **Response Measure**:
    - **Onboarding throughput**: All 500 merchants successfully registered per minute
    - **Provisioning time** for new instances < 5 minutes
    - **No increase** in average transaction latency beyond 10%

Scenario SC2

- **Source**: Business decision to expand to multiple regions (e.g., EU, APAC)
- **Stimulus**: Launch in a new region with local data center and currency support
- **Artifact**: Global routing, replicated databases
- **Environment**: Multiregional production environment
- **Response**: New region becomes operational without major architectural rework; localized payment methods integrated
- **Response Measure**:
    - **Time to stand up new region** < 2 weeks
    - **New region latency** < 250ms (95th percentile) for local users

---

5. Maintainability

Scenario M1

- **Source**: Developer team merges new code for a subscription billing feature
- **Stimulus**: Continuous integration system runs automated tests and code quality checks
- **Artifact**: Code repository, build pipeline, deployment scripts
- **Environment**: Test environment mimicking production configuration
- **Response**: The system automatically builds, tests, and flags any regressions or integration conflicts
- **Response Measure**:

    - **Build success rate** ≥ 95%
    - **Time to detect** and fix integration issues < 1 day
    - **Test coverage** for new feature > 80%

Scenario M2

- **Source**: Production incident requiring a hotfix
- **Stimulus**: Bug reported in the payment authorization flow causing some transactions to be incorrectly flagged
- **Artifact**: Payment microservice or monolithic payment module
- **Environment**: Production with ongoing transactions
- **Response**: A patch is developed, tested in staging, and deployed
- **Response Measure**:
    - **Mean Time to Recover (MTTR)** from bug report to fix in production < 4 hours
    - **No repeat failures** after patch

---

6. Reliability

Scenario R1

- **Source**: Partial component failure in the Payment Service's primary database
- **Stimulus**: A node in the database cluster crashes during peak usage
- **Artifact**: Payment Service, transaction data layer
- **Environment**: Production, normal transaction volume
- **Response**: System re-routes queries to remaining nodes, local caching or replicas handle read/write continuity
- **Response Measure**:
    - **Zero lost transactions** or data corruption
    - **Automatic failover time** ≤ 30 seconds

Scenario R2

- **Source**: Coding error introduced in a deployment
- **Stimulus**: The error causes some transactions to be marked as "completed" before they are fully processed
- **Artifact**: Transaction state machine, DB consistency
- **Environment**: Production environment, moderate load
- **Response**: The system detects data inconsistency and rolls back incorrect transactions or flags them for review
- **Response Measure**:
    - **Number of affected transactions** < 0.01%
    - **Time to reconcile**: Data or transaction states rectified within 60 minutes

Trade-off Activity

Quality Attribute	Monolith Architecture	Microservices Architecture
Performance	- **Pros**: In-process communication can be faster (no network overhead between components). - Cons : A single deployment can become a bottleneck under heavy load; performance issues in one module can affect the entire system.	- **Pros**: Each service can be optimized for performance with the best-suited technology stack, and horizontally scaled as needed. - Cons : Inter-service communication adds network overhead, which can introduce additional latency.
Reliability	- **Pros**: Simpler debugging since all components are in one place; fewer moving parts can mean fewer independent failure points. - Cons : A single point of failure if the monolith crashes, it can bring down the entire system.	- **Pros**: Fault isolation—a failure in one microservice does not necessarily crash the rest of the system. - Cons : More complex failure modes introduced by distributed systems (e.g., partial failures, cascading failures).
Scalability	- **Pros**: Straightforward to scale by running multiple copies of the entire monolith (vertical scaling or “big box” servers).	- **Pros**: Granular scaling—services can scale independently based on demand (e.g., Payment Service might need more

	- Cons : You must scale everything together, even if only one module needs more capacity. Overprovisioning is common.	instances, while Merchant Onboarding stays minimal). - Cons : Operational overhead to manage and orchestrate multiple services.
Availability	- **Pros**: With proper replication/failover, a monolith can still achieve high availability. - Cons : Downtime for one component typically means downtime for the entire application; rolling updates are trickier.	- **Pros**: High availability can be improved by distributing services across multiple zones or regions; partial updates can be deployed independently. - Cons : Requires more sophisticated DevOps for service discovery, load balancing, and failover.
Security	- **Pros**: Fewer network endpoints (everything is internal), potentially simpler to secure at the perimeter. - Cons : Larger attack surface _within_ the codebase if cardholder data is spread throughout; entire codebase might be in PCI scope.	- **Pros**: Can isolate sensitive components (e.g., Payment Service) in a restricted environment, reducing PCI scope. - Cons : Many more network interfaces between microservices can increase the external “attack surface” if not carefully secured.
Maintainability	- **Pros**: Easier to start and understand (one repo, single deployment). - Cons : As the codebase grows, modules become tightly coupled; changes can have wide-ranging impacts, slowing development.	- **Pros**: Smaller, more focused codebases per service; teams can iterate independently and deploy more frequently. - Cons : Complexities in versioning APIs, dealing with inter-service compatibility, and debugging distributed transactions.