.NET System Design Cheat Sheet

Layered Architecture

- Presentation Layer: UI, API Controllers.
- Application Layer: Orchestrates use cases, validation.
- Domain Layer: Business rules, entities, value objects.
- Infrastructure Layer: DB, file system, external services.

Clean Architecture

- Core domain has no dependency on infrastructure.
- Use interfaces (ports) and adapters (implementations).
- Enables testability, separation of concerns, and maintainability.

Microservices

- Decompose into independently deployable services.
- Use service boundaries around business capabilities.
- Communicate via HTTP/gRPC or messaging (e.g., RabbitMQ, Kafka).
- Use API Gateway for routing, caching, authentication.

CQRS (Command Query Responsibility Segregation)

- Separate reads (queries) from writes (commands).
- Improves scalability and performance for read-heavy systems.
- Often paired with Event Sourcing for audit trails.

Scalability and Resilience

- Use caching (MemoryCache, Redis).
- Load balancing, retries (Polly), circuit breakers.
- Async messaging with queues (Azure Service Bus, SQS).
- Horizontal scaling with containers or serverless.

API Design

- Use RESTful principles with HTTP status codes.
- Validate input at boundaries.
- Version APIs, document with Swagger/OpenAPI.
- Secure with OAuth2, API keys, or JWTs.

Data Access Patterns

- Use Repository + Unit of Work for abstraction.
- Prefer async EF Core methods for non-blocking I/O.
- Use Dapper for performance-critical reads.

Observability

.NET System Design Cheat Sheet

- Centralized logging (Serilog, ELK, or Seq).
- Distributed tracing (OpenTelemetry, App Insights).
- Metrics and alerts with Prometheus/Grafana or CloudWatch.