System Design Concerns based on Product Stages

Concern	Rapid Prototyping	Minimum Viable Product (MVP)	Production-Ready System	System at Scale	Distributed Enterprise System
Goal	Validate feasibility & core idea	Build a functional product	Ensure reliability & maintainability	Handle high traffic efficiently	Ensure global availability, security, and resilience
Architecture	Monolithic, hardcoded logic	Modular monolith with Separation of Concerns	Modular monolith or microservices	Microservices, service-oriented architecture (SOA)	Fully distributed system with microservices and event-driven architecture
Tech Stack	Fastest-to-develop tech (e.g., Firebase, SQLite, Express.js)	Standard frameworks (Node.js, Flask, Spring Boot)	Well-tested frameworks with CI/CD	Kubernetes, containerization, cloud-native services	Multi-cloud, hybrid cloud, edge computing
Database	In-memory DBs (SQLite, JSON files)	Single relational DB (PostgreSQL, MySQL) or MongoDB or CouchDB	Optimized DB queries, indexing, caching	Sharding, replication, NoSQL (MongoDB, DynamoDB)	Globally distributed databases (Google Spanner, CockroachDB)
Scalability	Not a concern	Basic horizontal scaling	Load balancing & replication	Auto-scaling, CDN, caching layers	Multi-region, active-active setup
Fault Tolerance	No redundancy	Minimal error handling	Graceful error handling, circuit breakers	Redundant services, failover mechanisms	High availability (99.99% uptime), disaster recovery
Security	Minimal (if any)	Basic authentication (JWT, OAuth)	Role-based access, encrypted storage	Threat detection, Distributed Denial of Service (DDoS) protection, compliance	Zero Trust security, identity management, GDPR compliance
Testing & Deployment	Manual testing, minimal automation	Unit tests, Manual Deployment	Automated tests, CI/CD pipelines	Full CI/CD, blue-green deployments	Canary releases, observability-driven deployment
Monitoring & Logging	Debug prints, basic logs	Basic logging, crash reporting	Structured logs, performance monitoring	Centralized logging, distributed tracing (Jaeger, OpenTelemetry)	AI-driven observability, predictive analytics
Cost Efficiency	Low cost, quick iteration	Budget-conscious, cloud-based	Optimized infra usage, reserved instances	Auto-scaling, cost monitoring	Cost-efficient hybrid-cloud strategies
Data Processing	Local/in-memory	Basic CRUD operations	Optimized queries, indexing	Event-driven processing, batch jobs	Stream processing, real-time analytics
Inter-service Communication	None or direct function calls	Basic API endpoints (REST/GraphQL)	API Gateway, message queues (RabbitMQ, Kafka)	Service mesh (Istio, Linkerd)	Event-driven, asynchronous microservices