Architecture: Fundamental organization of a system, embodied in its components, their relationships to each other & the environment, and principles guiding its design & evaluation

ISO quality requirements: Functionality, Performance efficiency, Compatibility, Usability, Reliability, Security, Maintainability, Portability

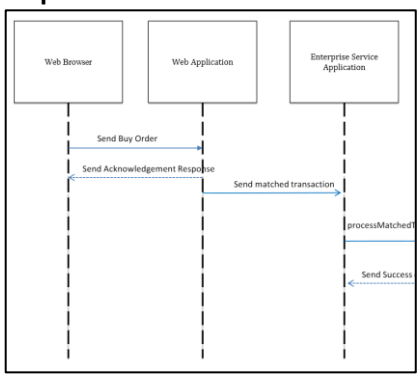
Scheduling algos: Round robin, priority, FCFS, shortest job first, shortest remaining time next, guaranteed scheduling

Demilitarized zone: Separates internal network from external network

Proxies: Reverse proxy hides identity of web server, forward proxy hides identity of client

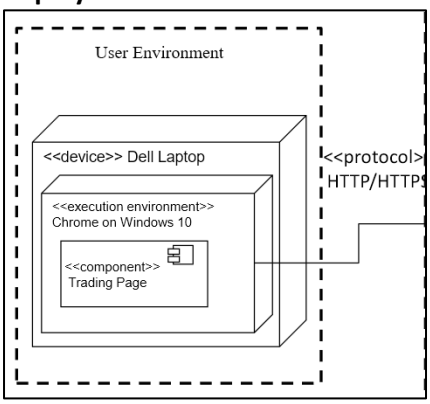
Diagrams

Sequence



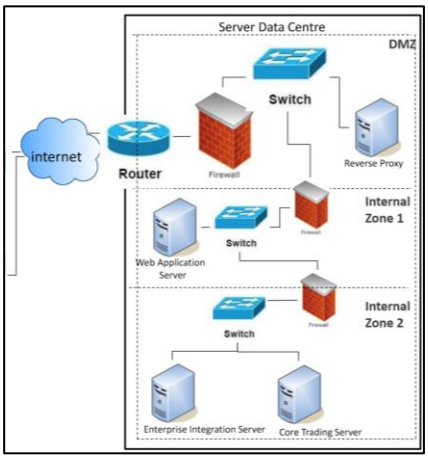
Components: sync msg, async msg, return msg

Deployment



Components: Location, node, artifact, communication path

Network



Components: Grouping, node, communication path

Networking and security

ARP	Link layer protocol to find MAC address of a node within the subnet, given its IP address
IP	Network layer protocol, allows for subnetting with CIDR notation
DNS	Application layer protocol to query the IP address of a domain name. Servers can be local(router) or authoritative(NS)
TCP	Transport layer protocol that provides reliable data transfer
SSL/TLS	Transport layer protocol that adds a layer of security through encryption (key exchange)
HTTP	Application layer, allows GET/POST/PUT/DELETE/PATCH
Threat	Potential danger to information
Vulnerability	Weakness within a system that can cause a breach/violation
Asset	Anything of value to the organization
Control	Safeguards to prevent/minimize loss
Risk	Probability of a threat exploiting a vulnerability and impacting an asset
Threat ex.	SQL injection, HTTP tampering, packet sniffing, DNS spoofing, MITM, HTTPS downgrade

Security Architecture design principles

Mitigate	Isolate components based on security levels, protect data in transit, storage & renewal, design for backups, limit attack surfaces, fail secure/safe, defense-in-depth, secure the weakest link
Detect	Monitor for malicious traffic, collect logs, use tools (OWASP ZAP)
Recovery	Data and system recovery methods
Firewall	Can be stateless/stateful, sets inbound/outbound rules for packet filtering – one table required for each

Rule no.	Interface	Src Addr	Src port	Dst Addr	Dst port	Action
#	eth0	X.X.X.X	<port(s)>	X.X.X.X	<port(s)>	ALLOW/DENY

Maintainability: Modularity, Analyzability, Testability, Reusability, Modifiability

Layered archi	Logical separation of software components – layers are dependent on those below and independent of those above. Can be open(bypassable) or closed(non-bypassable)+(Easy to debug, separation of responsibilities, reusability) –(Performance overhead, redundant code)
Client-server	Client sends requests, server returns responses. Can be strict client-server or p2p
Tiered archi	Tiers are treated as "layers", but components physically separate. Each tier is a server for its caller and client for its callee
CAP theorem	Consistency, availability, tolerance to partitions

Maintainability – Development Strategy	
Activity diagram	Illustrates various stages in the software development strategy, (e.g. develop -> test -> compile -> deploy to staging -> SIT & UAT -> deploy to production)
CloudFormation	Comprises description, metadata, parameters, mappings, conditions, resources and outputs
Grouping	Break up a template into smaller ones based on ownership and responsibility
Enterprise integration patterns	
File transfer	Files transferred in batch mode with defined format, protocol, frequency and server architecture. (+) universal integration, simple, high abstraction, (-) error processing, data sync, data-only(no metadata), one-way communication
Database integration	Supports batch or record/txn mode, guarantees atomicity, consistency, isolation, durability (ACID), allows shared/replicated DB configs
Interface integration	Screen/web scraping presentation layer, (+)minimal changes (-)consistency, formatting, preprocessing, speed
Messaging	Handles transfer of records/txns and can be centralized/decentralized. Define file format, selectivity, persistence(whether messages are stored on disk), durability(whether messages are still received when subscriber is not running) and exception handling. Define model(point-point/pubsub) and communication (polling/event-driven, sync/async)
APIs	Define functions, input/output params and exception handling. Expose function through a protocol. SOAP defines methods in XML, uses WSDL interfaces to expose functionality. REST organizes API into resources, and defines operations using HTTP methods
Brokering	Perform transformations on data payloads from one endpoint to another. (+)Can link any integration pattern, (-) incurs performance overhead
Architectures: Thin client, thick client, native, monolithic, services. Consider tradeoffs in speed, security, scalability	
Availability	
Tradeoff	Availability (operational & accessible) vs reliability (specified functions, conditions and time frames)
Redundancy	Vertical redundancy achieved by running multiple artifacts in a single node, horizontal redundancy achieved by running multiple nodes
Clustering	Active-active vs active-passive: How redundant nodes are configured to handle traffic Failure detection: Can be inbound(pinging) or outbound(heartbeat) Failover: Process of redirecting traffic/workload after a machine fails. Can be client-based, done by load balancer, DNS or virtual IPs
Replication	Sessions: Can be replicated through DB/cache, client, or in-memory Database: Can be done in master-slave (where slave acts as a backup w read permissions) or master-master (where both DBs can write). Important to implement precedence rules, primary key segregation or domain-specific differences
Design techniques	Separation of concern: Loose coupling between critical & non-critical components Fault tolerance: Ensure that errors do not terminate a service but cause it to enter a degraded state Parallel vs series: Having services in series always increases failure rate
AWS archi	ELBs to provide health-checks, security, TLS, balancing, elastic IP for virtual IP configs, route 53 for as an authoritative DNS for availability cross regions
Performance	
Metrics	Time behaviour, resource utilization, capacity
Techniques	Load balancing, parallel execution, caching and pre-fetching
AWS autoscaling	(AWS)Use ELB, autoscaling and cloudwatch to configure EC2 instances to scale out and scale in according to traffic with step adjustments according to CPU usage
SOLID principles and design patterns	
Single responsibility	A class should have one & only one reason to change
Open-closed	Open for extension, closed for modification
Liskov substitution	Derived classes must be able to substitute their base classes
Interface segregation	Make client-specific interfaces
Dependency inversion	Depend on abstractions, not concrete implementations. Dependencies should be injected into classes that require them
Singleton	Create 1 private, static instance of the class and provide global access to all other classes

Builder	Create classes in a step-by-step process of construction with a builder object
Factory	Define an interface for creating an object but let classes decide
Adapter	Convert existing interface into another interface that clients/users expect
Façade	Provide an interface to a set of interfaces that make the subsystem easier to use
Chain of Responsibility	Pass requests along a chain of objects (linked list) to avoid excessive coupling between sender and receiver
Observer	Create a one-many dependency between observers and subject, so that changes in subject are shared with all observers (push model)