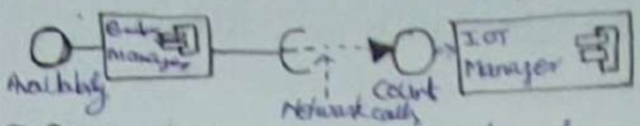
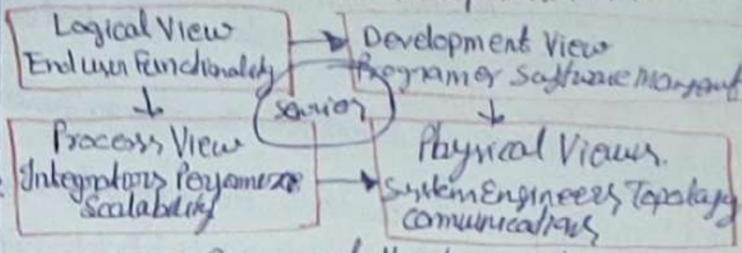
① VILAL ALI - 2020900023 END SEM-24

UML - ① Dependency class A to class B
② Association (has-a) - Class A offers class B.
Aggregation and composition
③ Generalization (Is-a) Class A is kind of C.B
DESIGN SMELL: Abstraction Missing
Imperative Abstraction, Duplicate Abstraction
② - Deficient, Leaky, Unexploited.
③ - Modularization Broken, Insufficient, Hublike.
④ - Hierarchy Missing, wide, cyclic, Deep
CODE SMELL: Long Method, Long list Parameter, Premitive Obsession, conditional complexity, Divergent changes, Feature Envy. Five main Categ.
① Blooters, OOA, change Preventers, Dispensable
Data Clumps, Lazy class, Temp Fields, Middle Man.
CODE METRICS: $V(G) = e - n + 2P$. e = edge, n = nodes, P = comp
SIX O METRICS WMPC, DITNC, CBOC, Response of class, Lack of cohesion on methods.
DESIGN PATTERNS:
① Creational Patterns: Focus on creation mechanism, providing flexibility in creating objects in a manner suitable for the situation.
(i) Singleton Pattern - Ensures a class has only one instance and provide a global point of Access.
(ii) Factory Method Pattern: Defines Interface for creating an object. But lets subclasses decide which class Initiate.
② Structural Patterns: Structural patterns deal with object composition and class structure emphasizing how classes and objects can be combined to form larger pattern structure.
Adapter pattern, Decorator and Facade Pattern
③ Behavioral Patterns: Focus on communication and class structure collaboration between objects defining patterns of communication.
① ~~Adapter~~ Observer Patterns, Strategy, command Patterns.
SOFTWARE ARCHITECTURE: A collection of computational components together with a description of the interaction between these components - the connectors.
Elements (components and connectors:
① Three Types: Data, Processing and connecting elements. ② They form the foundational pieces of a software architecture
Form [Architectural Patterns/styles]
Rationale (Design Decisions). Software Architecture is the earliest model of the whole system.
- A set of components and connectors communicating.
- A set of Architecture design decisions.
- Focus on set of views and view points.
- Architectural styles.

① Components and Connectors:



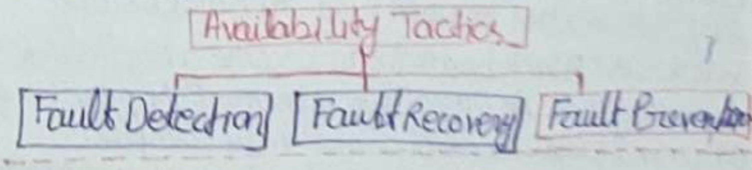
② Design Decisions - About what to choose.
Selected components/interfaces connectors.
Distribution/configurations of components/connectors expected behavior SA Styles, Patterns and Tactics HW/SW/Development and others views. Components Nestings and Subsystems NF attributes.
③ Architecture View and Viewpoints 4+1 view Model



SCENARIOS: Represent the different use cases.
Stakeholders: End-user, Developer.
Concerns: Understandability. Diagram: Use case diagrams.
Architecture Description

① SYSTEM QULITIES: ① Availability, ② Security ③ Performance ④ Modifiability, ⑤ Testability ⑥ Usability ⑦ Sustainability
- ARCHITECTURAL TACTICS:
① Availability. Downtime per year.

90% 36.5 Days. 0.365 × 24 = 8.46 hrs.
99% 3.65 Days
99.9% 8.46 Days. hrs.
99.99% 52.34 minutes
99.999% 5.26 minutes.
99.9999% 32 seconds.

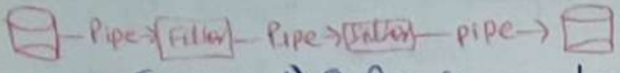


- PERFORMANCE:-
① Resources Demand, Resource Management Resource Arbitration.
- Security: System providing:- ① Confidentiality ② Integrity ③ Availability ④ Non-repudiation ⑤ Assurance ⑥ Auditing.
- Modifiability is About the cost of change
Localize changes, Prevent Ripple Effects
Deffer Binding Time.
- Testability: Demonstrate its fault
① Manage Input/Outputs, ② Internal Monitoring.
- Usability: Design Time, Runtime.

H-203, CS

② SOFTWARE ARCHITECTURAL STYLES

1- Pipe and Filter Pattern- Intuitions

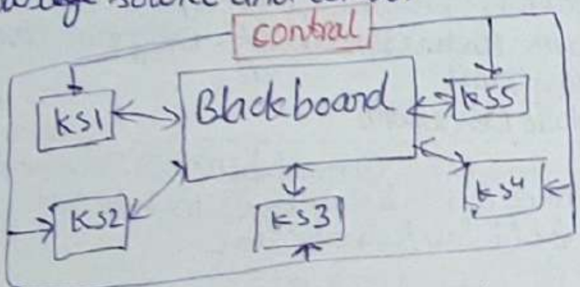


① Filter(Component) ② Pipe (connectors)

① Transformation Data from input to output. Con execute concurrently, incrementally transform.
② Single source for input and single target for output
③ More like a linear sequence of actions ⇒ pipelines.
① Not good for interactive system.
② Large number of filters can add substantial overhead

③ BlackBoard Pattern :

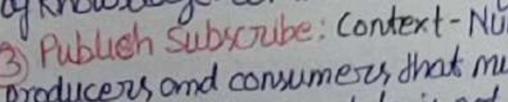
Context: Open problem domain with various partial solutions.
Problem: The partial solutions need to be integrated
Solutions: Decompose the software into blackboard, knowledge source and control.



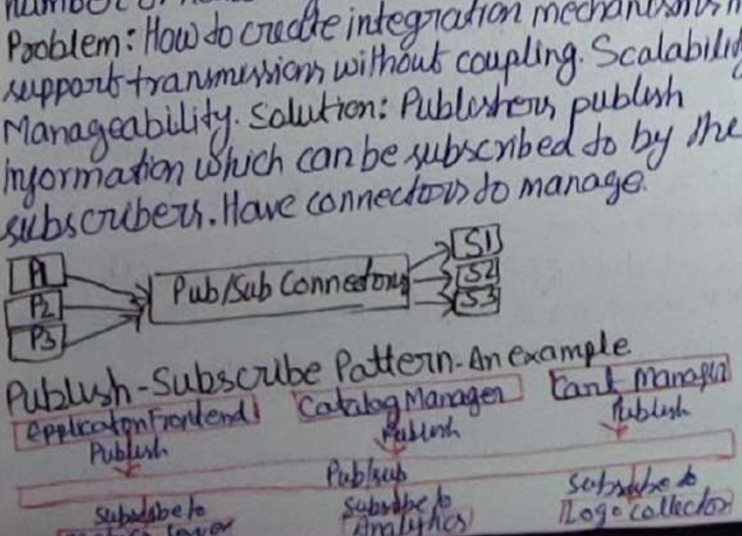
1. The Blackboard: Global repository containing input data and partial solutions.
2. Knowledge Source (KS) - Seprate and Independent components. Contains the knowledge required to solve
3- Controller - Component managing course of problem solving (eg. manage ks)

Constraints: No Direct communications among the ks. Any Interactions happens via the blackboard.
Weakness: Blackboard can become a bottleneck too many (KS). Difficult to determine the partitioning of knowledge. Control can be very complex.

③ Publish Subscribe: Context - Number of independent producers and consumers that must interact. The number or nature of data is not fixed.
Problem: How to create integration mechanisms that support transmissions without coupling. Scalability Manageability. Solution: Publishers publish information which can be subscribed to by the subscribers. Have connectors to manage.

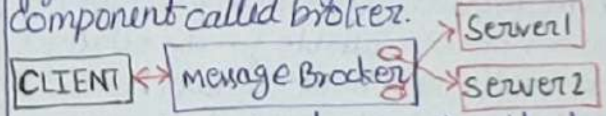


Publish-Subscribe Pattern- An example



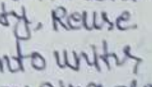
Architectural Elements: Publisher: Components that produces messages/events.
Subscriber: Components that consume the message events produced by publisher. Pub-Sub connector component that has announce and listen roles for publishers and subscribers.
Constraints: All components are conneted to a connector (Bus or a Component). Restriction on which components can listen to what. A component may be both a publisher and subscriber.
Weakness: May Increase latency, Can have a Negative impact. Less control on ordering of message. Delivery of message is not quranteed.

BROKER: Context: Many systems are collection of distributed programs. They need to exchange information and be available. Problem: How to do structure a distributed system such that service users need not worry about location of providers. Availability, Interoperability
Solution: Seperate user of funtionalities from provider of functionalities using an intermediatory component called broker.

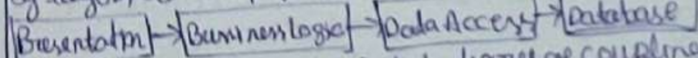


Constraints: ① Client can only attach to a brocker ② Server can only attach to a brocker.
Weakness ① Brocker can results in performance bottelneck (latency). ② Brocker can be a single point failure ③ Can be subject to security attack

LAYERED: Context: Develop and evolve points partition of system independently. Promote seperation of concerns. Problem: Modules can be developed and evolved seperately with little interaction. Modifiability, Portability, Reuse.
Solutions:- Devide the software into units called layers. Each layer is a grouping of modules
Layer - Kind of a module.
Relation - Allowed to use.
Constraint: Every peice of software is exactly allocted one Layer.

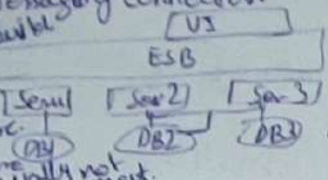


There are atleast two layers (often more)
③ Allowed to use relations should be acyclic.
Weakness: ① Performance bottlenecks ② The addition of layers adds up-front cost and complexity



- Monolothic Approach: ① High degree of coupling - everyone needs to knows everything ② change cycle and Bug fix can take weeks. Modifiability. ③ Adding new feature can be challenging - Extensibility.
④ Separation of concerns via components with inherent coupling - Modularity ⑤ Scaling System implies scaling the whole stack - Scalability ⑥ Limited by the language of choice. ⑦ Database centralized Addition or modification is costly process.

③ Service Oriented Patterns: ① Service Providers - Components that provides 1 or more services through defined interfaces ② Service Consumers: Invoke services directly or through intermediary ③ ESB - Intermediary components that can route and transform messages ④ Service Registry - Providers can register services consumers can discover services. ⑤ Orchestration Server: Coordinates interaction between consumers and providers based on languages
Connectors: SOAP Connectors - SOAP Protocol for synchronous communications over HTTP ② Rest Connectors ③ Asynchronous messaging connector.

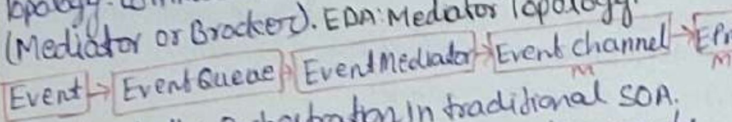


weakness: Complex to build Performance bottleneck due to middleware guarantees are usually not meet.

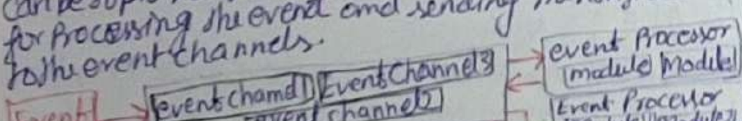
Constraints - Service consumers are connected to providers (ESB or other intermediatory component may be used

Microservice Key Advantage: Scaling is Easy: Scale only the required microservices. Heterogeneity - Each microservice can be developed in diff technologies. Resilience - Only specific microservice goes down

EDA - ① Independent components asynchronously emit and received events communicated over event busses ② Produce, Detect and consume events ③ Highly decoupled event component - Minimal Amount of coupling (topics queue names, etc). Elements: Component event producer, event consumers. Connectors: event bus Topology: Communication via the event bus or link only (Mediator or Brocker). EDA: Mediator Topology.

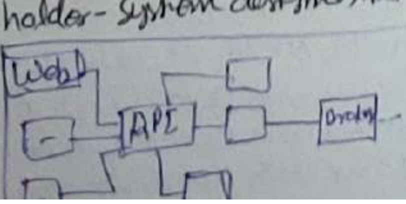


• Similar to the Orchestration in traditional SOA.
• Two key events - Initial and Processing events
Components: Event Queue - Responsible to transfer events to ev mediator. E Mediator - Orchestrates the processing of events to accomplish the overall functionality. event channel - Topics or queues to which events are ingested by mediator (Kafka topic). Event Processor - Implements the business logic. Can be fine grain or Coarse grained. Advice keep it to one functionality.
EDA BROKER Topology: Similar to the choreography in traditional SOA. • Two main types of components: ① Broker - Consist of all the event channels for event processing can be topics or queues. ② Event Processor - Responsible for Processing the event and sending the notifications to the event channels.



Advantage: High performance, High scalability. Ease of Deployment, Ease of modifications/Evolved easily.
Disadvantage: Remote process, availability - Liveliness of a consumer. ① Lack of Responsiveness, ③ Broker or mediator failures. Testing can be tedious. Development can be complex
EDA eg: Kafka, nest, spark Apache, Real time Dashboard

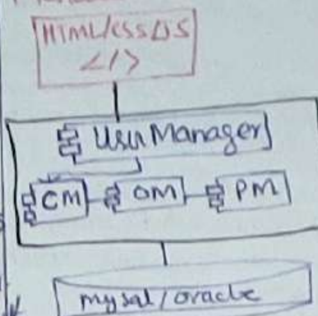
Physical View: Stack holder - System designer, Admin
Concerns - Performance, Scalability, Availability.
Deployment Diagram.



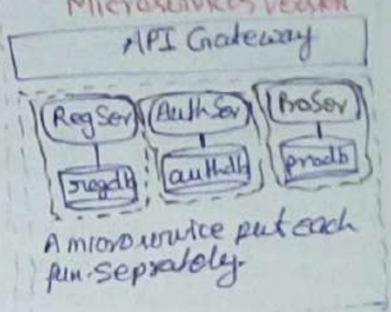
Use case 2 - E-Commerce:
Component or service that make such a subsystem:
Analytics Subsystem: Data Collection, Processing, Storage Analysis, and visualization.
Quality Requirement: Scalability, Reliability, real-time processing, Flexibility to accomodate varying load
Describe the topology: pipe and filter pattern is rec Components Include Event Producers, Event Consumers event Busses or Brocker, Message queues and business logic
Opinion on scalability and performance of the system. The observer pattern

Symbols and Use case: Microservices (Fine Grained) Monolithic (Single Unit). SOA (Coarse-grained)

Monolithic Version.



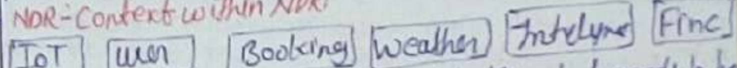
Microservices Version



A microservice put each fun. separately.

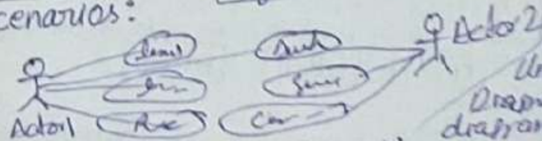
Microservices - How to design:
- Follow the Principle of bounded contexts: Identify real different contexts inside the main domain.
- Ensure loose coupling: - Minimized coupling between microservices. Easy to change and deploy one without
- Maintain high cohesion - Bundle one end to end feature
Promote robustness and reliability.

NDR - Context within NDR:



Hidden and shared Models: - Identify what needs to be shared. ② Same things may have different meaning in different context. - Microservices should never be chatty - Adds to performance issues, Lack of cohesion. Modules and services in NDR.

Scenarios: (Diagrams)



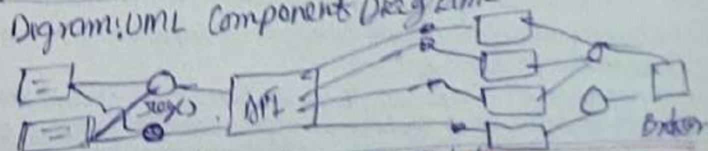
End user Developers. Concerns - Understandability Diagram - use case diagram.

Logical View / UML class Diagram:

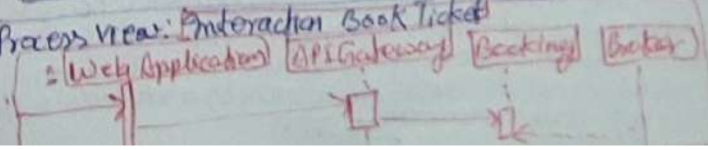
Stackholder - Developers.
Concerns - functionality

Development View: Stakeholders, Developers, manager. Concern - Organization, reuse, portability

Diagram: UML Component Diagram



Process View: Interaction Book Ticket



USE CASE-1 Q1- The potential issue in the System architecture could be RMS is a built as a monolithic system. This type of architecture typically leads to tight coupling between different components making it difficult to debug, change part of the codebase independently and release new features without impact the entire system.

Q2- The suggested refactoring for the function 'xyz' with multiple parameter due to There are many parameter resulting in Long Parameter List smell and one way to refactor is to "Preserve Whole Object". This approach involves passing a single object (eg. a Pilot Object) containing all these details instead of passing each parameter separately which can simplify the method signature and improve code maintainability.

Q3- The code smell observed in the described class structure. This is an example of a Hierarchy smell. Having a parent class (x) with multiple child classes (a, b, c) each containing a shared composit relationship (xy) suggested a potential issues with the design hierarch, which could lead to increased complexity and maintenance challenge over time.

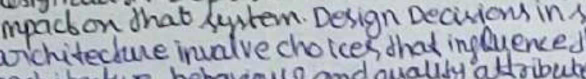
Q4- The creation of (x) object within the (xy) class violates the information expert principle. According to this principle, responsibility for creating object should lie with the class that has the most information required to properly initialise these objects.

Q5- The Development view, as part of the 4+1 Views model, will primarily help. It will help the developers and it will describe the structural composition through the component diagrams. This view provides developers with insight into the modular structure of the system using component Diagrams which illustrate how different software components interact and organize within the system.

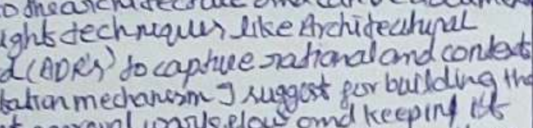
Q6- The best description of design decisions is Architectural design decision represent that can have a significant impact on that system. Design Decisions in software architecture involve choices that influence the system architecture, behaviour and quality attributes. They are fundamental to the architecture and can be documented using light weight techniques like Architectural Decisions Record (ADR's) to capture rational and contexts.

Q7- The implementation mechanism I suggest for building the driver enrolment approval work flow and keeping it automated is Use pipes and filter architectural patterns. This pattern suitable for sequentially processing a series of independent steps. Where each step can act as a filter processing.

Q9-(a)- BLACKBOARD PATTERN:

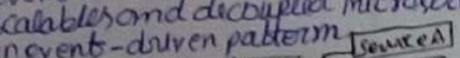


PIPE AND FILTER PATTERN: The pipe and filter pattern is well-suited for tasks that can be broken down into sequence of processing steps.



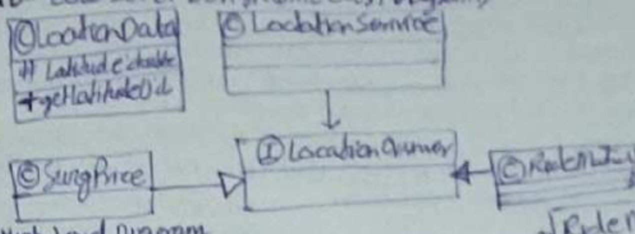
Q9b- Choosing between the Blackboard pattern and the pipe and Filter pattern for the Rideside Data analytics subsystem involves considering various quality attributes and their trade-offs. Lets examine both patterns in this context. Preferred Pattern. For the system data analytics subsystem I would lean towards the pipe and filter pattern based on the following considerations. ① Simplicity and Clarity ② Modularity and maintainability. ③ Scalability and Performance. Blackboard patterns offers flexibility and supports more complex collaborative problem solving.

Q10a- Components- Event Bus, Booking, User, Driver, Notification, Payments, Analytics. Q10-b Quality Requirement - Scalability, Reliability, Performance, Security, Resilience, Event Consistency. Q10C-Topology Diagram for Event-Driven - The chosen topology for the event-driven ride booking subsystem is based on a scalable and decoupled microservices architecture using an event-driven pattern.

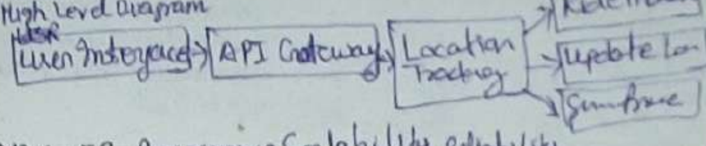


Q11a- Using the observer pattern in a monolithic architecture to handle real time location tracking for 1000 users per 2nd may present scalability and performance challenges.

11b- Low Level Design (UML class Diagram)

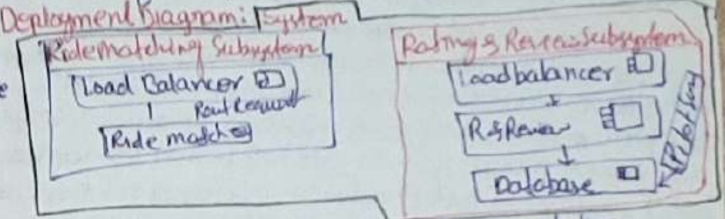


High Level Diagram



Q12 a- NFR- Performance, Scalability, Reliability.

b- Stakeholders and concerns: Passengers, Drivers, Operations Team management. Tactics and Patterns: Load Balancing and caching- Implement load balancing to distribute traffic and caching to improve performance.

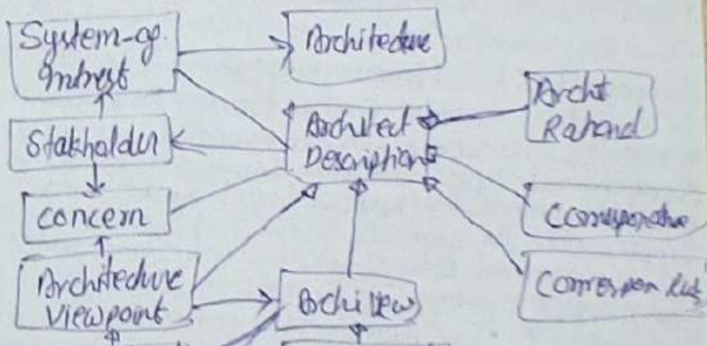


F- Design Patterns for Low-Level Implementation.

① Observer patterns: Used for real-time updates in the Ride

② Strategy Patterns- Employed Rating & Review system.

Architecture Description:



HSSM ⇒ $V = N\log_2(n)$.

Vocabulary $n = n1 + n2$, Program length $N = N1 + N2$

Volume $V = N\log_2(n)$

Operators (+, *, =, duble, int, final, return, {, }, (,)), $n1 = 11$

operands (calculateTotalCost, item1, item2, sum tax, number1, number2, total cost) = 8.

$N1 = (1, 1, 3, 2, 3, 1, 1, 1, 1, 1, 1) = 17$

$N2 = (1, 1, 1, 2, 2, 1, 1, 2) = 11$

$n = 19, N = 28, V = 28\lg(19) = 35.80$.

Feature Envy, Data clumps, Primitive obsession.

Using many Primitive data types.

Same data items together in many places.