K. J. Somaiya College of Engineering, Mumbai-77

Batch: A1	**Roll No.: 16010121045**
Experiment / assignment / tutorial No. 3	

TITLE : Identifying the Architectural Style of any B2B Software Applications

AIM: To visualize the architectural style of existing proprietary software based on the features of the application

Expected OUTCOME of Experiment:

CO 1. To design the architecture of software systems in various architectural styles and patterns.

Books/ Journals/ Websites referred:

1 "Software Architecture, Richard N Taylor etl, Wiley

2 www.google.com

Theory:

Definition

An architectural style is a named collection of architectural design decisions that are applicable in a given development context constrain architectural design decisions that are specific to a particular system within that context elicit beneficial qualities in each resulting system.

Benefits of Using Styles:

1. **Design reuse**
   Well-understood solutions applied to new problems
2. **Code reuse**
   Shared implementations of invariant aspects of a style
3. **Understand ability of system organization**
   A phrase such as "client-server" conveys a lot of information
4. **Interoperability**
   Supported by Style standardization
5. **Style-specific analyses**
   Enabled by the constrained design space
6. **Visualizations**

Style-specific depictions matching engineers' mental models

Style Analysis Dimensions:

1. What is the design vocabulary? Component and connector types
2. What are the allowable structural patterns?
3. What is the underlying computational model?
4. What are the essential invariants of the style?
5. What are common examples of its use?
6. What are the (dis)advantages of using the style?
7. What are the style's specializations?

Some Common Styles:

- **Traditional, language-influenced styles**
  - Main program and subroutines
  - Object-oriented
- **Layered**
  - Virtual machines
  - Client-server
- **Data-flow styles**
  - Batch sequential
  - Pipe and filter
- **Shared memory**
  - Blackboard
  - Rule based
- **Interpreter**
  - Interpreter
  - Mobile code
- **Implicit invocation**
  - Event-based
  - Publish-subscribe
- **Peer-to-peer**
- **"Derived" styles**

  - C2
  - CORBA

Proprietary S/W Application Name : DropBox

Software Application Design decisions:

File Storage and Retrieval

- **Object Storage & Distributed File Systems**
  - Should be able to save files in object storage systems.
  - Should support distributed file systems for large-scale storage.
- **Data Redundancy**
  - Mechanisms to tackle and minimize data redundancy.
- **Backup and Recovery**
  - Should support comprehensive backup and recovery processes.
- **Synchronization**
  - Should be able to synchronize files across devices and platforms.

User Interface Requirements

- **Cross-Platform Compatibility**
  - Should work seamlessly across multiple operating systems (e.g., Windows, macOS, Linux).
- **Integration with OS File Explorers**
  - Should integrate with native file explorers for ease of access.

Scalability Requirements

Distributed System Architecture

- **Load Balancing Techniques**
  - Implement load balancing to evenly distribute network traffic.
- **Data Partitioning and Replication**
  - Utilize data partitioning and replication for efficient data storage and access.
- **Sharding Strategies**
  - Apply sharding strategies to handle large datasets effectively.

Performance Optimization

- **Caching Mechanisms**
  - Use caching mechanisms (e.g., in-memory, distributed) for faster data retrieval.
- **Asynchronous Processing**
  - Support asynchronous processing to improve performance under high loads.

Scalability Testing

- **Performance Benchmarks and Load Testing**
  - Conduct performance benchmarks and load testing to assess system capabilities.
- **Capacity Planning**
  - Engage in capacity planning to ensure the system can scale as needed.

Security and Privacy

- **End-to-End Encryption**
  - Provide end-to-end encryption options to protect user data.

User Experience (UX) Requirements

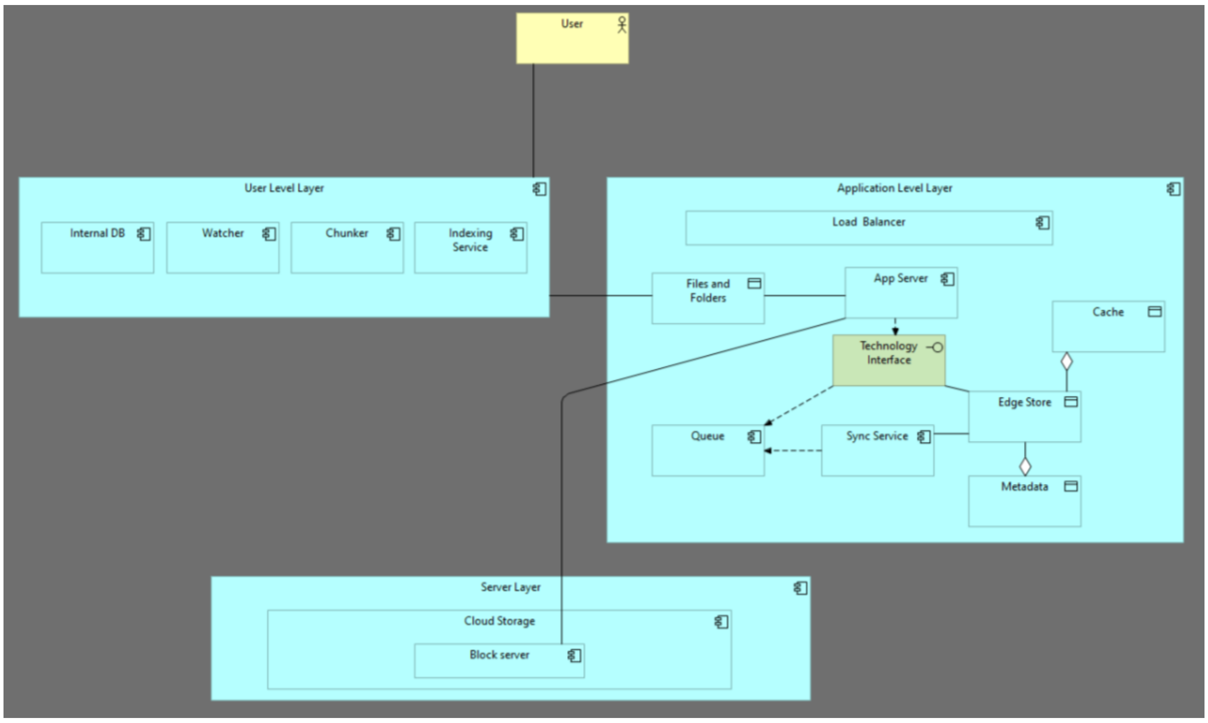
- **File Preview and Sharing**

- Enable users to preview files and share them easily.
- **Search Functionality**
    - Implement robust search functionality for easy file retrieval.
- **Version Control**
    - Support version control to manage file revisions effectively.

Business Model

- **Tiered Pricing Models**
    - Offer tiered pricing models to cater to different user needs and budgets.

Architectural Model:



Architectural Styles Identified & Why that style:

The Layered architecture style was selected for this system due to its advantages:

Modularity and Maintainability: It divides the system into distinct layers, each with well-defined responsibilities. This modularization enhances code organization, making the system easier to understand, modify, and maintain.
Scalability: The layered approach allows for independent scaling of individual layers based on performance requirements. For example, if the database layer becomes a bottleneck, it can be scaled horizontally without affecting other layers.
Reusability: Components within each layer can often be reused in other systems, promoting code efficiency and reducing development time.
Flexibility: The layered structure provides flexibility in terms of technology choices. Different technologies can be used within each layer, allowing for best-of-breed solutions and adaptability to changing requirements.

Explain the components and connectors of for the selected application
Ans:
Components:

1. User Level Layer:

- **Chunker:** Breaks large files into smaller chunks for efficient transmission and storage.
- **Watcher:** Monitors files for changes and triggers updates.
- **Indexing Service:** Creates and maintains an index of files for efficient searching and retrieval.
- **Internal DB**: used for fast retrieval of data.

2. Application Level Layer:

- **Load Balancer:** Distributes incoming requests across multiple servers to improve performance and scalability.
- **Cache:** Stores frequently accessed data locally to reduce latency and improve performance.
- **Metadata Service:** Manages metadata associated with files, such as file names, timestamps, and attributes.
- **App Server:** Handles incoming user requests, processes data, and interacts with other components.
- **Sync Service:** Synchronizes files across different devices and locations.
- **Queue:** Manages tasks for asynchronous processing, such as file synchronization or background jobs.
- **Edge Store:** Stores data closer to users for reduced latency and improved performance.

3. Server Layer:

- **Cloud Storage:** Provides scalable and reliable storage for files and data.

Connectors:

1. Association Connector

- **User and User Level Layer:** This is correct because a user is associated with the User Level Layer, which provides the user interface and functionality.
- **User Level Layer and Folders:** This is correct because the User Level Layer interacts with and manages folders, which are data objects.
- **Folders and App Server:** This is correct because the App Server processes requests related to folders, such as creating, deleting, or accessing them.
- **Technology Interface and Edge Store:** This is correct as the Technology Interface defines the interactions between the Edge Store and other components.

2. Aggregation Connector

- **Edge Store and Cache:** This is correct because the Edge Store might contain or aggregate a cache for improved performance.
- **Edge Store and Metadata:** This is correct as the Edge Store includes metadata as part of its data storage functionality.

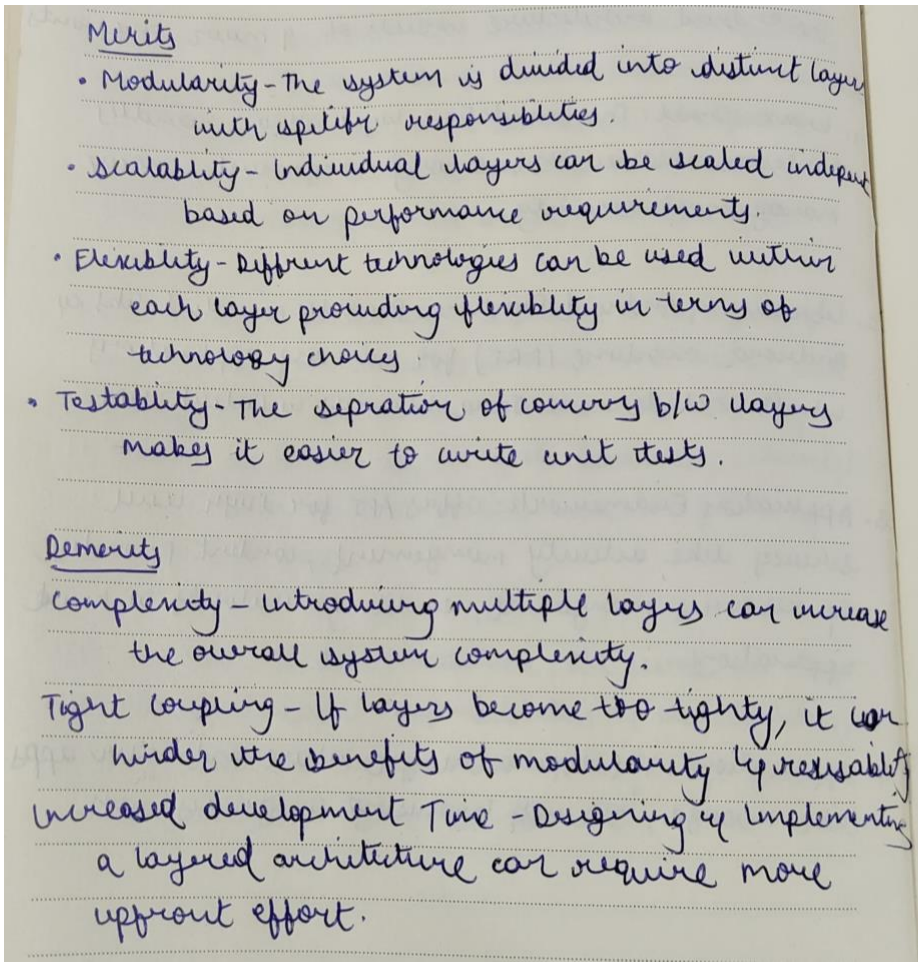
3. Flow Connector

- **App Server and Technology Interface:** This is correct as the App Server uses a technology interface to interact with the Edge Store or other components.
- **Technology Interface and Queue:** This is correct as the Technology Interface is used to send tasks to the queue for asynchronous processing.
- **Sync Service and Queue:** This is correct because the Sync Service likely receives tasks from the queue for synchronization operations.

<u>Post Lab Descriptive Questions</u>

1. **Discuss the merits and demerits of the style of the selected application.**

Ans:



<u>Merits</u>

- Modularity - The system is divided into distinct layers with specific responsibilities.
- Scalablity - Individual layers can be scaled independ based on performance requirements.
- Flexiblity - Diffrent technologies can be used within each layer providing flexiblity in terms of technology choices.
- Testablity - The sepration of concerns b/w layers makes it easier to write unit tests.

<u>Demerits</u>

Complexity - Introducing multiple layers can increase the overall system complexity.

Tight coupling - If layers become too tighty, it can hinder the benefits of modularity & reusablity

Increased development Time - Designing & Implementing a layered architecture can require more upfront effort.

Date: ______________ **Signature of faculty in-charge**