COMSATS University Islamabad, Abbottabad Campus

Lab Assignment 02

NAME: ***HASHAM KHALID***

REGISTRATION NO: ***FA22-BSE-138***

SUBJECT: ***SDA***

SUBMITTED T0: ***Sir Mukhtiar Zamin***

DATE: ***1/02/2025***

Five Major Architectural Problems

1. Scalability Issues

Problem Description:
Scalability refers to a system's ability to handle increased loads, whether due to a growing number of users, transactions, or data volume. Many older systems, especially those built on monolithic architectures, struggle with scalability because they rely on a single codebase and centralized database.

Causes:

- Poorly designed architecture that cannot handle increasing traffic.
- Tight coupling between system components.
- Single-point bottlenecks that slow down the entire application.
- Limited horizontal scaling options (e.g., adding more servers).

Effect on Software:

- System performance degrades under high user loads.
- Frequent crashes and downtime during peak usage.
- Difficulty in adding new features without impacting existing functionality.
- Increased operational costs due to inefficient resource usage.

Solution:

To solve scalability issues, **Microservices Architecture** is often adopted. In this approach:

- The system is broken into smaller, independent services.
- Each microservice handles a specific functionality and can be scaled independently.
- Load balancers are introduced to distribute traffic evenly.

Example: An e-commerce platform migrating from a single database structure to independent services for products, orders, and user accounts.

2. Monolithic Architecture Bottlenecks

Problem Description:
Monolithic architecture refers to building a software system as a single, unified application. While this approach works for small systems, it creates problems as the application grows in size and complexity.

Causes:

- Large codebases that become hard to understand and maintain.
- Dependency issues where a small change in one module affects the entire system.
- Deployment challenges, as every update requires redeploying the entire application.
- Limited flexibility to use different technologies for different modules.

Effect on Software:

- Difficulty in maintaining and understanding large codebases.
- Slow deployment cycles, as minor changes require full application redeployment.
- Dependency issues cause cascading failures across modules.
- Limited flexibility to adopt new technologies or tools.

Solution:

The solution lies in **Modular or Microservices Architecture**, where:

- The application is broken down into independent modules/services.
- Each service runs independently and communicates via APIs.
- CI/CD pipelines are used for seamless deployment.

Example: A banking system where transaction services, account services, and user management are decoupled into separate services.

3. Poor Data Management and Integration

Problem Description:

As systems grow, the volume of data and the need for smooth integration between different databases and services become critical. Poor data management leads to inefficiencies, data duplication, and slow performance.

Causes:

- Inefficient database design or normalization.
- Poor data synchronization between different services.
- Lack of clear data governance policies.
- Data silos where data is isolated in separate systems.

Effect on Software:

- Data duplication and inconsistencies across the system.
- Slow response times due to inefficient database queries.
- Poor integration between different system components or third-party services.
- Lack of real-time data processing leads to outdated information being displayed to users.

Solution:

The adoption of **Event-Driven Architecture (EDA)** or tools like **Apache Kafka** allows:

- Real-time data processing.
- Better integration across services.
- Event streaming for smooth communication between data sources.

***Example*: ** A retail chain implementing event-driven data pipelines to ensure inventory data remains consistent across warehouses.

4. Security Vulnerabilities

Problem Description:

Security remains one of the biggest concerns in software architecture. Outdated or poorly designed architectures often leave systems exposed to data breaches, hacking attempts, and unauthorized access.

Causes:

- Weak authentication and authorization mechanisms.
- Outdated security patches.
- Poorly encrypted data storage.
- Lack of compliance with security standards (e.g., GDPR, HIPAA).

Effect on Software:

- Increased risk of data breaches and hacking incidents.
- Loss of user trust and legal consequences for non-compliance with security standards (e.g., GDPR, HIPAA).
- Data leaks leading to financial and reputational damage.
- Patching vulnerabilities becomes expensive and ineffective in older systems.

Solution:

Implementing **Zero Trust Architecture (ZTA)** and regular **security audits** ensures:

- Every access request is validated, regardless of its origin.
- Data encryption is enforced both in transit and at rest.
- Regular vulnerability assessments are conducted.

Example: A healthcare application upgrading its authentication system to use multi-factor authentication (MFA).

5. Legacy System Modernization

Problem Description:
Many organizations still rely on legacy systems-built decades ago. These systems often lack compatibility with modern technologies and become expensive to maintain.

Causes:

- Dependency on outdated frameworks and languages.
- Difficulty in integrating new technologies.
- Lack of proper documentation for old systems.
- High cost of migrating to new platforms.

Effect on Software:

- High maintenance costs and increased technical debt.
- Poor compatibility with modern tools, APIs, and cloud technologies.
- Dependency on outdated technologies or frameworks no longer supported.
- Limited ability to introduce new features without significant rework.

Solution:

A gradual **refactoring approach** along with **containerization technologies (Docker, Kubernetes)** is used to modernize these systems.

- Applications are moved to container-based environments.
- Old modules are rewritten using modern programming languages.
- Continuous Integration/Continuous Deployment (CI/CD) practices are implemented.

Example: A government agency shifting its legacy payroll system to a cloud-based microservices architecture.