Name: Toheed Ahmed Qureshi

Reg No: FA21-BSE-156

Submitted to: Sir Mukhtiar Zamin

Subject: Software Design & Architecture

Introduction:

Software architecture is the backbone of any software system which define its structure components and interaction with the user. A good architecture have scalability performance maintainabaility and much more factors. Software systems often face significant architectural challenges that can alter their effectiveness, reliability, and longevity.

Findings:

The finding from analysis of software architecture challenges reveals some major problems that are listed below:

- Software systems are increasingly complex, often leading to integration and scalability issues.
- Performance bottlenecks and maintainability concerns are common in legacy systems.
- Security vulnerabilities and fault tolerance remain critical areas for improvement.
- Rapidly evolving business requirements demand more flexible and adaptive architectures.
- Over-engineering and technical debt are prevalent in fast-paced development environments.
-

Problems:

- Scalability Limitations
- Deployment Challenges

- Complexity and Maintenance
- Performance Bottlenecks
- Security and Compliance
- Complexity in Monitoring and Debugging
- Lack of Modularity
- Data Consistency and Integrity

I choose the platfrom of Ebay which is the ecommerce website launched first for UK. Ebay started with a monolithic architecture to manage its services and backend. This design is sufficient for the early customers of ebay.

Some key issues they face are:

1. **Scalability Challenges**:
   - The monolithic structure meant all functionalities (e.g., user management, product listings, search, payments) were tightly coupled.
   - Scaling required increasing resources for the entire system, even if only specific functionalities needed more capacity (e.g., search during peak holiday seasons).
2. **Deployment Bottlenecks**:
   - A change in one part of the system required testing and redeploying the entire application, slowing down release cycles.
   - This risked downtime and made it harder to roll out new features quickly.
3. **Fault Isolation**:
   - Bugs or failures in one component could potentially crash the entire system due to the lack of clear boundaries between functionalities.
4. **Development Complexity**:
   - Adding new features became increasingly difficult as developers had to navigate a large and tightly integrated codebase.
5. **Resource Inefficiency**:
   - Resource allocation could not be optimized, as all components were scaled together, regardless of individual usage.

Competitors of Ebay:

Amazon was one of the largest competitors and a significant threat to eBay. While eBay primarily focused on auctions during its early years, Amazon concentrated on direct sales and later expanded into third-party marketplaces.

Architecture of Amazon:

During the period when eBay struggled with scalability issues due to its monolithic architecture, Amazon had already adopted a **microservices architecture**, which played a crucial role in its ability to scale efficiently and capture the market. By breaking down its platform into small, independent services, Amazon was able to scale different parts of its system (such as product management, user accounts, and payments) independently based on demand. This architectural choice allowed Amazon to handle massive traffic spikes, such as during holiday shopping seasons.

Solutions:

To address these challenges, eBay embarked on a major architectural overhaul by adopting a **service-oriented architecture (SOA)**. It is a **software design approach** where applications are built by combining small, reusable, and loosely coupled services. Each service represents a specific business capability and can function independently while interacting with other services through well-defined interfaces and protocols.

Decoupling Functionalities:

- The monolithic application was broken down into smaller, independent services, each responsible for a specific business capability (e.g., user management, search, payment processing, inventory management).
- Each service was designed to function autonomously, with clear boundaries and interfaces (often RESTful APIs).

Independent Scaling:

- Services could be scaled individually based on their specific workloads. For example:
    - Search services could be scaled during high traffic times.
    - Payment services could be scaled based on transaction volumes.
- This approach optimized resource utilization and ensured high availability.

Improved Fault Tolerance:

- Failures in one service were contained and did not affect other services. For instance, a bug in the recommendation engine would not disrupt payment processing.

Faster Deployment Cycles:

- Teams could work on, test, and deploy their services independently without affecting the entire system.
- This enabled eBay to roll out updates and new features more frequently.

Technology Agnosticism:

- Services could be developed using different technologies or languages, based on the most suitable tools for each task (e.g., Java for search, Node.js for user-facing APIs).

Introduction of Middleware:

- eBay implemented middleware for inter-service communication to ensure smooth data flow and consistency across services.
- Messaging queues and service buses were used to handle asynchronous operations and improve reliability.

Problem Taken:

Deployment Bottleneck:

Preventive measures:

- Modularization:
  Refactor the application into **modules or layers** where each module corresponds to a specific functionality such as user management, order processing. This will help me to changes in one module are less likely to impact others.
  Allows for easier testing and faster deployment of updates within specific modules.

Code Example:

```
public class Application {

  public void registerUser(String username, String email) {

    System.out.println("Registering user: " + username + " with email: " + email);

  }


  public void processOrder(int userId, String orderDetails) {

    System.out.println("Processing order for user " + userId + ": " + orderDetails);

  }


  public void generateReport() {

    System.out.println("Generating report...");

  }


  public static void main(String[] args) {

    Application app = new Application();
```

```
        app.registerUser("Toheed", "Toheed@gmail.com");

        app.processOrder(1, "Laptop");

        app.generateReport();

    }

}
```

Step 1: Create Modules

Each functionality is placed in its own class with a clear responsibility.

```
// UserManager Module

public class UserManager {

    public void registerUser(String username, String email) {

        System.out.println("Registering user: " + username + " with email: " + email);

    }

}


// OrderProcessor Module

public class OrderProcessor {

    public void processOrder(int userId, String orderDetails) {

        System.out.println("Processing order for user " + userId + ": " + orderDetails);

    }

}


// ReportGenerator Module

public class ReportGenerator {

    public void generateReport() {

        System.out.println("Generating report...");

    }

}
```

Step 2: Create an Application Controller

The controller acts as the orchestrator for interactions between modules.

```
public class ApplicationController {

  private final UserManager userManager;

  private final OrderProcessor orderProcessor;

  private final ReportGenerator reportGenerator;


  public ApplicationController() {

    this.userManager = new UserManager();

    this.orderProcessor = new OrderProcessor();

    this.reportGenerator = new ReportGenerator();

  }


  public void handleUserRegistration(String username, String email) {

    userManager.registerUser(username, email);

  }


  public void handleOrderProcessing(int userId, String orderDetails) {

    orderProcessor.processOrder(userId, orderDetails);

  }


  public void handleReportGeneration() {

    reportGenerator.generateReport();

  }

}
```

Step 3: Use the Controller in the Main Application

The controller simplifies the main application logic by delegating tasks to specific modules.

java

```
public class Application {
  public static void main(String[] args) {
    ApplicationController appController = new ApplicationController();

    // Register a user
    appController.handleUserRegistration("Toheed", "Toheed@gamil.com");

    // Process an order
    appController.handleOrderProcessing(1, "Laptop");

    // Generate a report
    appController.handleReportGeneration();
  }
}
```

References:

10 Lessons from Microservices Failures and How to Overcome Them » Engineering Bolt ⚡

Issues Faced by eBay: eBay_Architecture_Study.pdf