**Case Study :9**

**Task 1: What is Security concern for the product team?**

Product teams need to incorporate security concerns into their product plans, not as an ancillary service, but as a core feature of their product. Why? For one digital, Product teams need to incorporate security concerns into their product plans, not as an ancillary service, but as a core feature of their product. Digital insecurity has become normalized. Each week brings news of new attacks and new vulnerabilities. This is a little doubt that such attacks and vulnerabilities will inexorably follow Moore’s Law and continue to grow exponentially. And for another, consumers, businesses, and government agencies are becoming increasingly concerned about the safety and privacy of networks, transactions, and data.

**Task 2: Which application architecture pattern should use to develop new system or to maintain current existing system**?

An architecture pattern is a general, reusable solution to a commonly occurring problem in software architecture within a given context. Architecture patterns are similar to a software design pattern but have a broader scope.

In this article, I will be briefly explaining the following 10 common architecture patterns with their usage, pros and cons.

* Layered pattern
* Model-view-controller pattern
* Client-server pattern
* Master-slave pattern
* Pipe-filter pattern
* Broker pattern
* Peer-to-peer pattern
* Event-bus pattern
* Blackboard pattern
* Interpreter pattern
* **Layered architecture pattern**

The most common architecture pattern is the layered architecture pattern. Layered architecture patterns are n-tiered patterns where the components are organized in horizontal layers. This is the traditional method for designing most software and is meant to be self-independent. This means that all the components are interconnected but do not depend on each other. Each layer of the layered architecture pattern has a specific role and responsibility within the application.

The 4 layers of general information system are:

-Presentation layer (UI layer)

-Application layer (service layer)

-Business logic layer (domain layer)

-Data access layer (persistence layer)

* **Usage** (General desktop applications and E-commerce web applications)
* **Model-view-controller architecture pattern**

This pattern also known as MVC pattern, divides an interactive application into three parts as:

-**model**\_\_ contains the core functionality and data

-**view**\_\_ displays the information to the user

-**controller**\_\_ handles the input from the user

* **Usage**

**-** Architecturefor World Wide Web applications in major programming languages.

-wed frameworks such as Django, Rails.



**Task 3: Why should product team use enterprise application architecture pattern (tier/layer)?**

* An enterprise applications architecture describes the behavior of applications used in business, focused on how they interact with each other and with users.
* The applications architecture is specified on the basis of business and functional requirements.
* Applications architecture tries to ensure the suite of applications being used by an organization to create the composite architecture is scalable, reliable, available and manageable.
* Applications architecture defines how multiple applications are poised to work together. It is different from software architecture, which deals with technical design of how a system built.
* In applications management, applications are mapped to business functions and process as well as costs, functional quality and technical quality in order to assess the value provided.

**Task 4: Write down advantages and disadvantage of Tier/Layer architecture pattern?**

**Advantage of Tier/layer architecture**

* Layered architecture increases flexibility, maintainability, and scalability
* Layered architecture enables teams to work on different parts of the application parallel with minimal dependencies on other teams.
* Layered architecture enables develop loosely coupled systems.
* Different components of the application can be independently deployed, maintained, and updated on different time schedules.
* **Simplicity**: The concept of layered architecture is easy to learn and implement.
* **Consistency**: The layers along with the overall code organization is consistent across all the layered projects.
* **Reusability**: Components are reusable
* **Faster development**: Because of division of work web designer does presentation, software engineer does logic, DB admin does data model.

**Disadvantages of Tier/layer architecture**

* **Lack of inbuilt scalability**: The principles of layered architecture hinders the growth of your project as it does not help to scale your project.

You are required to find a key to organize it on your own.

* **Hidden use cases**: It is difficult to determine the use cases of your project simple checking the code organization.
* **No dependency inversion**: The dependencies are direct and conceptually changes into essential higher layers form a low-level infrastructure layer.
* **Development of user-interaction**: applications can sometime take longer if the layered prevents the use of user interface components that directly interact with the database.
* **Changes to lower level interfaces**: tend to percolate to higher levels, especially, if the related layered approach is used.