

## Assignment - 1

①

Date \_\_\_\_\_  
Page \_\_\_\_\_

Q-1

Ans:

To build our Application and its features <sup>we</sup> will be using microservices architecture.

- microservices Architecture is an approach in which a single application is composed of many loosely coupled and independently deployable smaller services.
- Some features of microservices are: highly maintainable and testable, loosely coupled, independently deployable, organized around business capability, owned by small team, flexibility and Agility, Isolation and Security, Better resource utilization, Adaptability to Growth.
- The microservices architecture enables the rapid, frequent and reliable delivery of large, complex applications. It also enables an organization to evolve its technology stack.
- For instance, if we want to change our product images and how they look. Then it shouldn't be a problem because all the different tasks over here are provided



as service. If you want to change the way the products image looks then you will have to change only this particular application right so by only changing this service your job will be done. you do not have to worry about the other services and you do not have to change anything over there so the main change you will have to be doing will be restricted to only this particular service so you save time and you save effort over here in case of microservices shopping application.

→ Smaller environments including web and mobile applications do not require such a robust communication layer and are easier to develop using a microservices architecture.

→ we cannot use monolithic architecture in this case as it becomes too large with time and hence difficult to manage. we need to re-deploy the whole application, even for a small change. As the size of application increases, its startup and deployment time also increases. for any new developer joining the project, it is very difficult to understand the logic of a large monolithic application even their responsibility is related to a single functionality.



Q-2 A hospital management Company wants to build a system to keep record of doctor / patient check in and another system for managing hospital parking lot. Analyse both the cases and suggest whether it a good idea to build a microservice or buy a commercial off the shelf software for each case.

Ans: If we consider building a microservice from the scratch then we have to keep some factors in our mind such as,  
Development Costs,  
Maintenance Costs,  
Scaling Costs.

→ If we consider purchasing a software then some factors to check are:  
Vendor reputation,  
Support,  
Integrations.

→ When it makes sense to build. There are 2 scenarios when opting to build is the best solution.

- The first is if your microservices make up the core functionality of your organization and are the key differentiator for your business. When it comes to technology, keeping the core of your product or service proprietary is always preferred.



- Secondly, it makes more sense to build if you are simply unable to find a product in the market that suits your need.
- When it makes sense to build, if the functionality you need is not part of your product core, and is therefore not a key differentiator in the market - like authentication or content management, software for example - it makes a whole lot more sense to buy. That's especially true if you come across a high-quality vendor that can easily integrate with your existing stack.
- For a parking lot management system, considering that reqs. are likely to be standard and less complex compared to a custom solution, buying a COTS software could be a more efficient and cost-effective option.
- In both cases, the decision should be made based on a careful analysis of the specific needs, available resources, tested level of customization, integration requirements and long-term maintenance considerations.



Q-3 Do we have to use only Java for implementing microservices? Justify your answer.

Ans:

No, we do not have to use only Java for implementing microservices. There are several good reasons to consider other languages and frameworks:

- Polyglot Architecture: one of the benefits of microservices is the ability to use different prog. languages based on what is best suited for each service. Enforcing a single language across all services loses this advantage.
- Leverage different lang. strengths:-
  - Different lang. have different strengths. for Example, node.js is a good for IO-heavy workloads, Go is good for building network services, Python has a lot of data science libraries etc. Using only Java would restrict tapping into these strengths.
- Avoid vendor lock-in:- Having microservices implemented in different languages prevents over-reliance on a single technology stack and avoids vendor lock-in.
- Developer Productivity:- Forcing all developers to use Java even if they have little experience with it may impact their productivity.



→ So, in summary, While Java is a popular choice, using multiple languages provides significant benefits like a better leveraging of language capabilities, developer productivity, avoidance of vendor lock-in and improved scalability and fault isolation. A Polyglot Architecture is preferred for microservice.

Q-4

What will be the output of the following code snippets. Explain the reason for the obtained outputs:

```
(1) function display() {
    var a = b = 10;
}
display();
console.log('b', typeof b === 'undefined');
console.log('a', typeof a === 'undefined');
```

Sol :-  
b false  
a true

→ The display() function declares a variable 'a' and assigns it the value of 'b'. Since 'b' is not declared, it becomes an implicit global variable.

→ When display() is called, 'b' is initialized to 10 and 'a' is also assigned the value 10.



- After `display()` exits, 'b' is still available in the global scope while 'a' was local to the `display()` function.
- The first `console.log` checks if 'b' is undefined, which evaluates to `false` since 'b' exists in the global scope.
- The second `console.log` checks if 'a' is undefined, which evaluates to `true` since 'a' was limited to the `display()` function scope.

\* (2) 'use strict';  
 function display() {  
   var a = b = 10;  
 }  
 display();  
 console.log('b', typeof b === 'undefined');  
 console.log('a', typeof a === 'undefined');

Sol:- ~~b false~~ Error: b is not defined.  
~~a true~~

- In the first code only 'a' is defined and 'b' is not 'b' so the output shows 'a' as `true` and 'b' as `false`.
- Here, 'use strict' is used. Strict mode makes it easier to write 'secure' JavaScript. Strict mode changes previously accepted bad syntax into real errors.
- In strict mode, any assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable or a non-existing object will throw an error.