🏠_Home_

🔗 Layard Architecture

Layered Architecture Interview Questions

1. What is software architecture?

2. Why do we need software architecture?

3. What is MVC.?

4. What is the difference between Layered and MVC architecture?

5. Describe the layers of Layered Architecture?

6. What are the rules which we should not violate while creating a software architecture?

7. What is cohesion?

8. What is loose Coupling, and why should we use it.?

9. How to apply loose Coupling for two Tightly Coupled classes.?

10. Explain what are boilerplate codes?

11. State disadvantages of having boilerplate codes?

12. What is Dependency Injection?

13. Explain about Dependency Injection methods.

14. What are Design Patterns.?

15. Explain about the design pattern which you used in the project.

16. Why did you use the Factory Design Pattern?

17. What is Facade?

18. Why did you use the Singleton Design Pattern?

19. Where did you use the Strategy Design Pattern and explain why?

20. Explain the responsibility of the DAO layer.

21. Explain the responsibility of the Business Layer.

22. Why did you add a SuperDAO interface in the DAO Layer?

23. Explain why you added the CrudDAO interface and why did you use generics inside it?

24. Why did you add a custom package to the DAO Layer and what's the use of interfaces inside it?

25. What is the responsibility of the Controller?

26. Explain why you create a CrudUtil (SQLUtil)?

27. Where did you handle transactions on this project?

28. What is the difference between Entity and DTO?

29. Why did you create a SuperBO in Business Layer? Explain.

30. Where can we write down a join query in your project?

31. Explain the data flow from Views to Data Layer.

32. What are the OOP concepts which you have used in layered architecture.?

33. Name a place where you used Inheritance.

34. Name a place where you used runtime polymorphism.

35. Name a place where you used encapsulation.

36. Name a place where you used compile time polymorphism.

37. What is the OOP concept which you used while applying loose coupling.?

38. Name a place where you used abstraction.

39. What are enums.?

40. What are the benefits of using layered architecture over MVC.?

1. Fundamental organization of a software system (මූලික සංවිධානයක්)

2. * increase the performance of the software
   * make it easy to fix bugs
   * Avoid complex codes

3. design architecture ekak..
   software eka model,view,controller widiyata bedala develop karana eka

4. * mvc architecture kiyanne architectural design pattern ekak
   * MVC wala design principals violate wenaw

   * layered architecture kiyanne mvc eka extend karala hadapu architectural design
     pattern ekak.
   * layered eke design principals violate wena eka awama karagnna puluwn

5. presentation layer □ *view*
   business layer / Service Layer □ *(Handle user data using Persistence layer and Data layer, Heart of the application)*
   persistence layer / Repository / DAO (Data Access Object) □ *query*
   data layer □ *Database*
6. 1.loose coupling
   2.dependency Injection
   3.less boilerplate codes
   4.high cohesion

7. As a unit, a class should include only relevant things. (Unit ekakata tiyenna one ekata adala dewal witri)
   Ex: Inside of the Customer Controller class has to include only relevant things of that.

8. * class ekk thawa class ekk matha directly depend wena ekka nathi kireema
   * To avoid loos coupling between classes.

9. Get all the methods inside the high level class and create an interface with those.Then we implement that interface to high level class and when we create object we use interface reference to assign the high level class object.

10. Repetitive and Complex Code

11. Complex repetitive codes. It's hard to understand or read the code. Hard to fix bugs

12. Concept of applying dependencies in a meaningful way.

13. 1.Property Injection - class එක create කරන වෙලාවෙම dependency inject කිරීම.
   2. Constructor Injection - Object එක create කරන වෙලාවෙම dependency inject කිරීම.
   3. Setter method Injection - Setter method එකක් trough dependency inject කිරීම.
   4. Interface through Injection - interface එකක් trough dependency inject කිරීම.

14. common solutions for common problems.

15. 01. singleton(DB connection, DAO Factory, BOFactory) // එකම object එක re-use වගෙ එක තවත්තන්න
02. factory (DAOFactory, BOFactory) // Object creation logic eka hide krnna.
03. facade (CrudDAO<T>) // Most common methods, එක තැනකට (interface එකකට) gather කිරීම. (for reusability)
04. strategy (DAO(CustomerDAO, ItemDAO, OrderDAO, OrderDetailsDAO))
// Unique methods Implement karanna. (for Runtime Logic selection)

16. factory - Object creation logic eka hide krnna.

17. facade - Gathering most common methods into a single interface to improve code reusability-CrudDAO

18. singleton - object ekak eka parak hadala eka re use karanna

19. Customer DAO,ItemDAO,OrderDAO,OrderDetailsDAO
* Unique methods Implement karanna. (for Runtime Logic selection)

20. data layer eke thiyana tables wlata query supply krna eka

21. user requirements walata anuwa business logic impliment krn eka

22. factory ekata common return type ekk denna

23. Code Reusability eka pawathwaganimat ha Boiler Plate codes walakwa ganimata

24. to maintain the loos coupling of DAO layer and also to supply unique methods (queries) for tables

25. view eka thula athi events tika manage kirimata

26. Code Reusability eka pawathwaganimat ha Boiler Plate codes walakwa ganimata

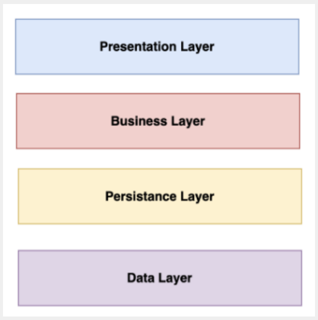
27. Business Layer

28. *Data layer eke thiyena table ha property gana coding level idea ekk ganna entity classes create karnwa
*user requirements walata anuwa data apata compatible format ekakta business layer ekat transfer karanna DTO classes use karanwa

29. BOFactory ekat most common return type ekk one nisa

30. QuaryDAO

31.



32. 1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism

33.

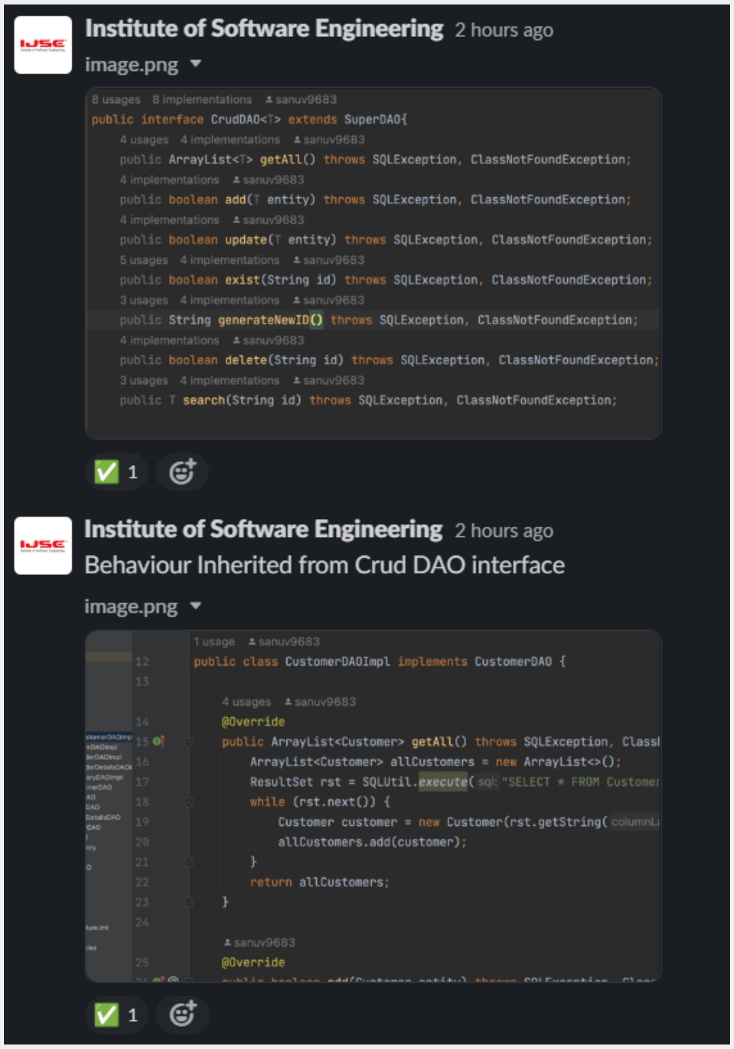

Institute of Software Engineering 2 hours ago

image.png

```
8 usages  8 implementations  sanuv9683
public interface CrudDAO<T> extends SuperDAO{
    4 usages  4 implementations  sanuv9683
    public ArrayList<T> getAll() throws SQLException, ClassNotFoundException;
    4 implementations  sanuv9683
    public boolean add(T entity) throws SQLException, ClassNotFoundException;
    4 implementations  sanuv9683
    public boolean update(T entity) throws SQLException, ClassNotFoundException;
    5 usages  4 implementations  sanuv9683
    public boolean exist(String id) throws SQLException, ClassNotFoundException;
    3 usages  4 implementations  sanuv9683
    public String generateNewID() throws SQLException, ClassNotFoundException;
    4 implementations  sanuv9683
    public boolean delete(String id) throws SQLException, ClassNotFoundException;
    3 usages  4 implementations  sanuv9683
    public T search(String id) throws SQLException, ClassNotFoundException;
```

✅ 1

Institute of Software Engineering 2 hours ago

Behaviour Inherited from Crud DAO interface

image.png

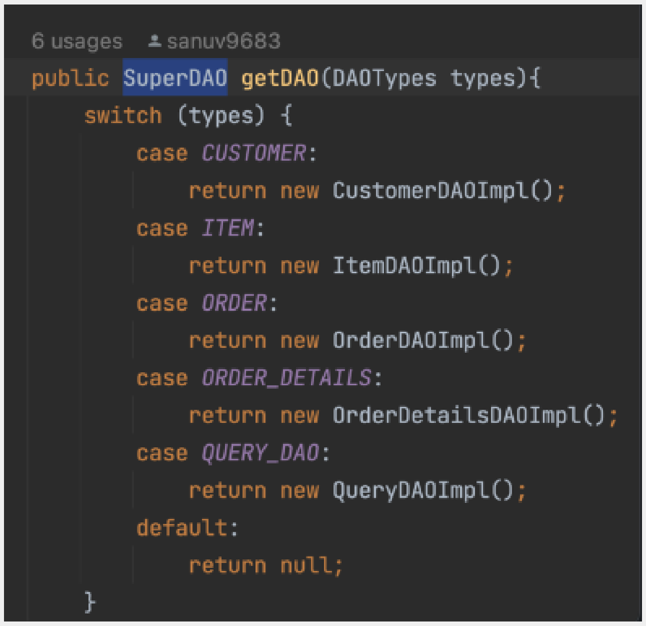
```
1 usage  sanuv9683
public class CustomerDAOImpl implements CustomerDAO {

    4 usages  sanuv9683
    @Override
    public ArrayList<Customer> getAll() throws SQLException, Classl
        ArrayList<Customer> allCustomers = new ArrayList<>();
        ResultSet rst = SQLUtil.execute( sql: "SELECT * FROM Customer
        while (rst.next()) {
            Customer customer = new Customer(rst.getString( columnL
            allCustomers.add(customer);
        }
        return allCustomers;
    }

    sanuv9683
    @Override
```

✅ 1

34.



```
6 usages  sanuv9683
public SuperDAO getDAO(DAOTypes types){
    switch (types) {
        case CUSTOMER:
            return new CustomerDAOImpl();
        case ITEM:
            return new ItemDAOImpl();
        case ORDER:
            return new OrderDAOImpl();
        case ORDER_DETAILS:
            return new OrderDetailsDAOImpl();
        case QUERY_DAO:
            return new QueryDAOImpl();
        default:
            return null;
    }
```

Runtime Polymorphism (Single Interface Many Forms)
SuperDAO is the Single Interface
Many Forms = CustomerDAOImpl, ItemDAOImpl, OrdersDAOImpl, OrderDetailsDAOImpl, QueryDAOImpl

35. DTO classes and ENTITY classes

36. DTO classes (Constructor overloading)

37. Inheritance and polymorphism(runtime polymorphism)

38. BO and DAO interfaces

39. An enum is a special "class" that represents a group of constants
(unchangeable variables, like final variables)

40. Clean Code
Less Complex
Easy to maintain
Easy to introduced new features
Easy to bug fix