Name :- Deepthi mashetti

Day 1 :

Computer Architecture

Softwares :- 1. System software

2. Application Software

System Software:

- This acts as an interface between the system and the applications.
- It is the platform that allows the various application software to run on the system.
- System Software is generally developed in low-level languages. This is so that the interaction between the software and hardware can be simplified and made more compatible.
- Is working is more automated. Once a system is turned on, the system software starts working
- The system software are installed at the time of installing the operating system. A computer device cannot work without its presence.
- It is an independent software. Once this is installed the computer will work.
- Since a device cannot work without a system software, the user has to have it installed in their devices.
- Example: System Software includes Android, Mac Operating system, MS Windows, etc.

. Application Software :

- This is designed directly from the user perspective.
- These are independent applications which can be download and installed in the system.

- Each application has a specific purpose and thus is developed with high-level languages so that the purpose can be fulfilled.
- User action is required to start application software. These applications can only be work when the user commands the system to do so.
- They have minimum involvement in the processing and functioning of the computer device.
- The application software can be installed as and when the user requires them.
- This is a dependent software. Applications can only be downloaded when the operating system is installed.
- These are designed to be user interactive, thus the application software can be removed as and when required by the user.
- Examples of Application Software includes Word Processor, games, media player,Device Drivers etc.

DAY 2 :

Architectural styles

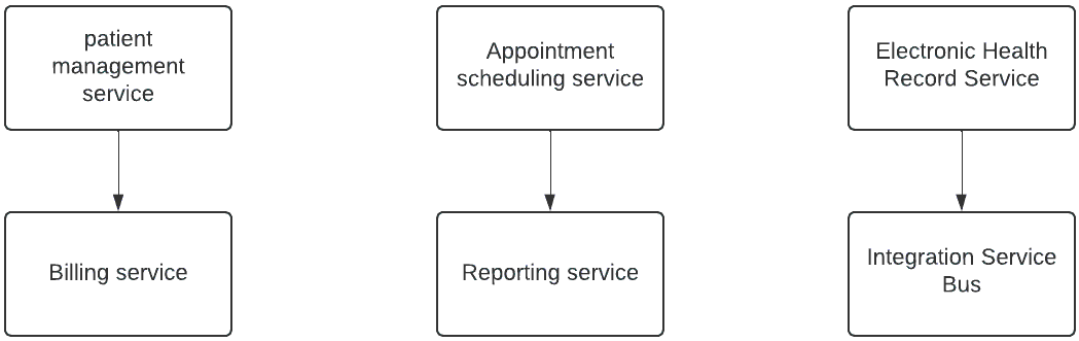
- Layered
- Client server
- Micro Services
- Event Drivers
- Service Oriented Architecture (SOA)

Service Oriented Architecture (SOA) :

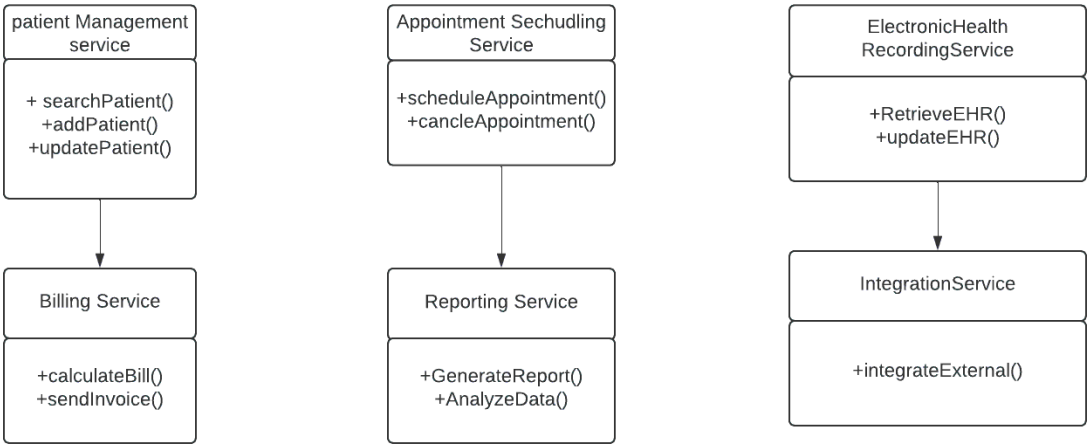
Case Study: Modernizing a Healthcare Information System with SOA.

https://pdfs.semanticscholar.org/a513/e1b39847faff37e34d6538d5e6459dde3925.pdf

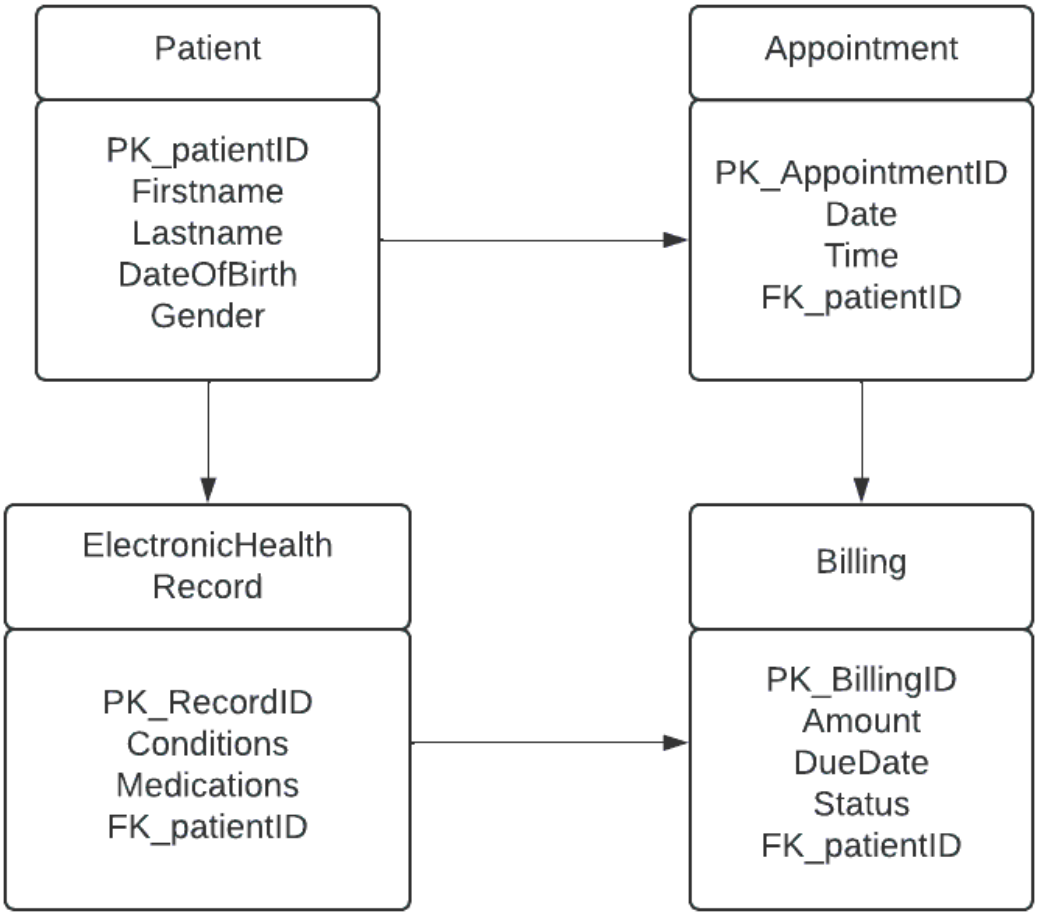
UML Component Diagram:-



UML Class Diagram :



SQL Diagram :



It is a huge collection of services in which services communicate with each other. This framework is used for designing a software system using a software architecture that views all components as services. The communication is provided by SOA generally used for data passing it could communicate two or more services and complete some activity. It provides various microservices as function which has following –

1. Loosely coupled

2. Reusable

3. Composable

4. Autonomic

5. Standardized

Advantages :-

- **Loosely coupled :** SOA promotes loose coupling between services, which means that services are not dependent on each other. This results in increased flexibility, scalability, and maintainability of the system.
- **Reusablilty** : Services are designed to be reusable, so they can be used in multiple applications.
- **Scalability:** SOA allows for easy scaling of applications because services can be added or removed without affecting the other parts of the system.
- **Flexibility:** SOA enables organizations to change and adapt to new business requirements easily, as services can be replaced or updated without affecting the rest of the system.
- **Interoperability:** SOA allows for the integration of different technologies and platforms, making it easier to connect different systems and applications.
- **Cost-effective:** SOA can be cost-effective in the long run because it reduces development time and makes it easier to maintain applications.

Disadvantage :-

- **Complexity:** SOA can be complex to design, implement, and manage, especially when dealing with large-scale systems and multiple services. This requires specialized skills and resources and can result in increased development time and costs**.**
- **Performance:** Services may introduce latency and overhead, which can negatively impact the performance of the system. This requires careful design and optimization of services and infrastructure.
- **Security:** Because services are accessible over the network, security becomes a concern, and additional measures must be taken to ensure that data is secure. This requires careful design and implementation of security measures.

- **Testing:** Testing services can be complex, and ensuring that all services work together correctly can be challenging. This requires specialized testing tools and techniques and can result in increased testing time and costs.
- **Governance:** Proper governance is required to ensure that services are designed, developed, and deployed according to established standards and best practices. This requires a comprehensive governance framework and a culture of compliance and accountability.

MicroService Architecture :

It takes large number of services and breaks down into small services or shareable components. It also called a monolith in which all functionality is placed into a single process. This approach is used for developing application and communication with different approaches which may write in different programming language and data storage. Here are some characteristic functions of MSA –

1. Business capabilities
2. Products
3. Smart End Point
4. Automation
5. Evolutionary

Advantage :-

- **Scalability:** Microservices can be easily scaled horizontally, allowing organizations to handle increased traffic and user requests without affecting the performance of the system
- **Agility:** Microservices enable organizations to quickly adapt to changing business requirements by allowing teams to develop and deploy services independently of each other.
- **Resilience:** Microservices are designed to be fault-tolerant, meaning that if one service fails, it does not affect the rest of the system.

- **Reusability:** Microservices can be reused across multiple applications, reducing development time and costs and promoting consistency across the organization.
- **Technology Diversity:** Microservices allow organizations to use different technologies and platforms for different services, enabling teams to choose the best technology for each service.
- **Continuous Delivery:** Microservices can be deployed and updated independently of each other, enabling continuous delivery and reducing time-to-market.
- **Easy Maintenance:** Microservices are easier to maintain than monolithic applications because each service is smaller and has fewer dependencies.
- **Increased Collaboration:** Microservices enable teams to work independently and collaborate effectively, resulting in increased productivity and faster development cycles.

Disadvantage :

- **Complexity:** Microservices introduce a higher level of complexity compared to monolithic applications. This complexity can result in increased development time and costs and requires specialized skills and resources.
- **Distributed System:** MSA involves building a distributed system, which can result in increased latency, network overhead, and communication complexity.
- **Data Management:** Managing data across different services can be challenging, and organizations must implement effective data management and synchronization strategies.
- **Testing:** Testing microservices can be complex and time-consuming, and organizations must implement effective testing strategies to ensure that all services work together correctly.
- **Deployment and Infrastructure Management:** Deploying and managing microservices can be challenging, especially when dealing with a large number of services. Organizations must have effective deployment and infrastructure management strategies in place.
- **Security:** Because microservices are distributed, security becomes a concern, and additional measures must be taken to ensure that data is secure.

- **Governance:** Proper governance is required to ensure that services are designed, developed, and deployed according to established standards and best practices.

DAY 3

SOA presentation

DAY 4

MVC (Model-View-Controller) and its variants

- The MVC full form, is a design pattern that divides your application into three primary parts: a model, a view, and a controller.
- The controller manages the flow of data between these two components.

Model :-

- The model part of the MVC architecture contains all relevant information about what should be stored in the database or other places where it would be needed later on during the development or testing of your codebase.

View :-

- The view part of the MVC architecture contains HTML code that describes how to display something on the screen which can then be accessed by controllers in order to respond to user requests and perform actions based on user input received through actions passed down from higher levels.

Controller :-

- The controller part of the MVC architecture is responsible for accepting requests from users and passing them on to appropriate views.
- The MVC controller also handles any tasks that need to be performed before or after displaying a view, such as validating input or storing data.

- The MVC controller is responsible for coordinating interactions between models and views.

MVC Features :-

- The MVC framework is well-suited for applications that involve complex interactions between multiple objects and views. Some of the MVC framework features are:
- Reduced complexity
- Increased testability and maintainability
- Better separation of concerns

The MVC pattern subsequently evolved, giving rise to variants such **as hierarchical model–view–controller (HMVC), model–view–adapter (MVA), model–view–presenter (MVP), model–view–viewmodel (MVVM),** and others that adapted MVC to different contexts.

model–view–presenter (MVP) :-

Model–view–presenter (MVP) is a derivation of the model–view–controller (MVC) architectural pattern, and is used mostly for building user interfaces. In MVP, the presenter assumes the functionality of the "middle-man". In MVP, all presentation logic is pushed to the presenter.

hierarchical model–view–controller (HMVC) :-

Hierarchical Model View Controller (HMVC) is an extension on the traditional Model view controller (MVC) architecture. It's main purpose is for use in web applications. The HMVC came about as a solution to scalability problems present in applications which used MVC**.**

model–view–adapter (MVA) :-

MVA is an architectural pattern that is also known as the “mediating controller MVC” or MVC with the mediator pattern applied. This pattern has all traffic pass through a “mediator” which is the Adapter in this case.

model–view–viewmodel (MVVM) :-

It is an architectural pattern which addresses the uneven scattering of functionality in an MVC by shifting the interface logic away from the View and presenting the view with bindings for direct access to model state or transforms/logic for model state.