1. **Dependency Injection**
   1. **Dependency Injection (DI)** is a design pattern used to implement Inversion of Control (IoC) between classes and their dependencies. It allows for better modularity, testability, and maintainability of code by decoupling the creation of dependencies from their usage.
   2. **Key Concepts**
      1. **Service:** A class that provides functionality to other classes.
      2. **Client:** A class that depends on the service.
      3. **Injector:** A class or framework that injects the service into the client.
      4. **Types of Dependency Injection**
2. **Constructor Injection -** In constructor injection, dependencies are provided through a class constructor.
3. **Property Injection -** dependencies are provided through public properties of the client class
4. **Method Injection -** dependencies are provided through method parameters
5. **MVC Architecture**
   1. **MVC (Model-View-Controller)** is a design pattern used for developing web applications. It separates an application into three main logical components: the Model, the View, and the Controller. This separation helps in organizing code, making it more modular, maintainable, and testable.
   2. **Components of MVC**
      1. **Model:**
         1. Represents the data and the business logic of the application.
         2. Interacts with the database and performs data-related operations.
         3. Notifies the View of any data changes.
      2. **View:**
         1. Represents the presentation layer of the application.
         2. Displays data to the user and sends user commands to the Controller.
         3. Updates the UI based on changes in the Model.
      3. **Controller:**
         1. Acts as an intermediary between the Model and the View.
         2. Handles user input and updates the Model.
         3. Selects the View to display based on user actions and Model state.



1. **Exception handling in MVC**
   1. Exception handling is a crucial aspect of building robust and reliable web applications. In ASP.NET MVC, there are several ways to handle exceptions effectively to ensure that your application can gracefully recover from errors and provide meaningful feedback to users.
   2. Methods
      1. Try-catch block
      2. Custom error page
      3. Global exception handling
      4. Logging exception details in files
2. **BIAN Architecture**
3. **API**
   1. **API (Application Programming Interface)** is a set of rules and protocols that allows different software applications to communicate with each other.
   2. APIs work by exposing endpoints that can be accessed by other applications or services. These endpoints are typically URLs that correspond to specific functions or data within the API. When an application makes a request to an API endpoint, the API processes the request, performs the necessary operations, and returns a response.
   3. **Key Components of an API**
   4. **Endpoint:**
      1. A specific URL where an API service is exposed.
      2. Each endpoint corresponds to a specific resource or action.
   5. **Request:**
      1. The message sent by the client to the API to perform an action or retrieve data.
      2. Includes the HTTP method (GET, POST, PUT, DELETE), headers, and optionally a body with data.
   6. **Response:**
      1. The message sent by the API back to the client.
      2. Includes status codes, headers, and optionally a body with data.
   7. **HTTP Methods:**
      1. **GET:** Retrieve data from the server.
      2. **POST:** Send data to the server to create a new resource.
      3. **PUT:** Update an existing resource on the server.
      4. **DELETE:** Remove a resource from the server.
   8. **Status Codes:**
      1. Standardized codes returned by the server to indicate the result of an API request.
      2. Examples: 200 (OK), 201 (Created), 400 (Bad Request), 404 (Not Found), 500 (Internal Server Error).
4. **Authentication vs Authorization**
   1. **Authentication**
      1. Authentication is required to ensure that the API is well secured to protect crucial data
      2. There are multiple ways to authenticate API
         1. Basic auth
         2. OAuth - Allows third-party applications to access user data without exposing credentials
         3. API key
         4. IDP token
   2. **Authorization**
      1. Ensure that the authenticated users have the necessary permission to access the specific resource or perform action
5. **IDP**
   1. **IDP (Identity Provider) Token** is a type of token issued by an Identity Provider (IDP) to authenticate and authorize users or applications. IDP tokens are commonly used in Single Sign-On (SSO) and OAuth 2.0/OpenID Connect (OIDC) scenarios to facilitate secure access to resources across multiple systems and applications.
   2. **Identity Provider**
      1. An IDP is a service that manages user identities and provides authentication services.
      2. Examples of IDPs include Google, Microsoft Azure AD, Okta, and Auth0.
   3. **IDP Token:**
      1. A token issued by the IDP after successful authentication.
      2. Types of IDP tokens include Access Tokens, ID Tokens, and Refresh Tokens.
   4. **Access Token:**
      1. A token used to access protected resources (APIs).
      2. Typically short-lived and contains information about the user's permissions.
6. **JWT**
   1. **JWT** is a compact, URL-safe token format used for securely transmitting information between parties as a JSON object.
   2. A JWT consists of three parts separated by dots (.):
      1. **Header:** Contains metadata about the token, such as the type of token and the signing algorithm.
      2. **Payload:** Contains the claims, which are statements about an entity (typically, the user) and additional data.
      3. **Signature:** Used to verify the authenticity of the token and ensure that the payload has not been tampered with.
7. **How does API communicate between domains**
   1. APIs (Application Programming Interfaces) between two domains communicate over the internet using standard protocols and formats. This communication typically involves HTTP/HTTPS requests and responses, and it can be facilitated by various mechanisms and technologies to ensure secure and efficient data exchange.
8. **Difference between .net core and MVC**
   1. **.NET Core** and **ASP.NET MVC** are both frameworks developed by Microsoft, but they serve different purposes and have distinct characteristics. Let's explore the differences between them.
   2. **.NET Core**
      1. **.NET Core** is a cross-platform, high-performance, open-source framework for building modern, cloud-based, and internet-connected applications. It is a general-purpose development platform maintained by Microsoft and the .NET community on GitHub.
   3. ASP.NET MVC
      1. **ASP.NET MVC** is a web application framework developed by Microsoft, based on the Model-View-Controller (MVC) architectural pattern. It is part of the ASP.NET framework and is used for building dynamic, data-driven web applications.
9. **Microservices**
   1. Building microservices in .NET involves using the .NET platform to create small, independently deployable services that work together to form a larger application. .NET provides a robust set of tools and frameworks to support microservices architecture, including ASP.NET Core, Docker, Kubernetes, and more.
   2. **Example:** Building a simple e-commerce application with Product Service, Order Service, and Inventory Service.
10. **Unit test case writing**
11. **DHP deployment**
    1. To onboard new API to DHP , environment.yml file is the first place to start with the changes. A new swimlane has to be identified here under the respective environment
    2. Swimlane is a logical group of all business applications that could be hosted on single or multiple EC2 instances
    3. The names for the swimlane and infrastructure definition should follow the naming convention rule
    4. Once swimlane defined then create/update specific yaml file depending on the security group in which Api needs to be hosted
    5. Define load balancer related configurations. Target group should be identified and health check endpoints also identified
    6. Once config are defined please raise PR. Once PR merged to master teamcity jobs corresponding to config defined will be created
    7. Next, provision the infrastructure as per config by running teamcity pipeline
    8. Also we need access to vault
    9. Finally deploy swimlane job will be triggered under the respective environment in teamcity
12. **Snyk**
    1. Snyk is a developer-first cloud-native security tool that finds and automatically fixes vulnerabilities in your code, open-source dependencies, containers, and infrastructure as code.
    2. Example : Password not masked
13. Sonarcube - checks the bad code
14. Docker – Verify if that is deployed to the latest version of the container
15. Redocly – Validate open API changes
16. Portal - Validate the code standards (naming conventions) based on the rules mentioned in the yaml file (gold class, silver class)
17. **What is stateful and stateless request give example**
    1. **Stateful** and **Stateless** requests are concepts used to describe how a server handles client interactions and maintains the state of a session. Understanding the difference between these two approaches is crucial for designing scalable and efficient web applications.
    2. **Stateless Requests**
       1. **Stateless Requests** do not retain any information about previous interactions. Each request from the client to the server is treated as an independent transaction, with no knowledge of prior requests. This approach simplifies the server design and makes it easier to scale the application.
       2. A common example of a stateless request is a RESTful API. Each API call is independent and contains all the necessary information to process the request.
       3. **Example: RESTful API Request**

GET /api/products/1 HTTP/1.1

Host: api.example.com

Authorization: Bearer your-access-token

* 1. **Stateful Requests**
     1. **Stateful Requests** retain information about the client's interactions over multiple requests. The server maintains the state of the session, allowing it to remember previous interactions and provide a more personalized experience.
     2. A common example of a stateful request is a traditional web application that uses server-side sessions to maintain user state.
     3. **Example: Stateful Web Application Request**

GET /dashboard HTTP/1.1

Host: www.example.com

Cookie: sessionId=abc123

1. **What is session and why to maintain**
   1. A **session** is a way to store information (state) about a user's interaction with a web application across multiple requests. It allows the server to remember the user's data and preferences as they navigate through the application. Sessions are typically used to maintain user-specific data, such as login status, shopping cart contents, and user preferences.
2. **Null pointer and object reference**
   1. **"Object reference not set to an instance of an object"**. This error occurs when you try to use an object reference that is null.
3. **Routing**
   1. **Routing** in ASP.NET is a mechanism that maps incoming HTTP requests to the appropriate controller actions. It is a fundamental feature of ASP.NET MVC and ASP.NET Core that enables the creation of clean, readable URLs and helps in organizing the application's structure.
4. **CRUD operations on http verb**
   1. CRUD operations (Create, Read, Update, Delete) are fundamental actions performed on data in a web application. These operations are typically mapped to HTTP verbs (methods) to interact with RESTful APIs. Each HTTP verb corresponds to a specific type of CRUD operation.
5. **Green Lane and Blue Lane**
   1. In the context of software deployment, **Green Lane** and **Blue Lane** are terms used to describe different deployment environments or lanes that facilitate continuous delivery and deployment of applications. These lanes are part of a strategy to ensure smooth and reliable deployments while minimizing downtime and reducing the risk of introducing errors into the production environment.
   2. Characteristics of green lane
      1. **Testing Ground**
      2. **Staging Environment**
      3. **Quality Assurance**
      4. **User Acceptance Testing (UAT)**
   3. Characteristics of Blue lane
      1. **Production Environment**
      2. **High Availability**
      3. **Monitoring and Maintenance**
6. **Middleware in .net ASP**
   1. **Middleware** in ASP.NET Core is a crucial component of the request processing pipeline. It is software that is assembled into an application pipeline to handle requests and responses. Each component in the pipeline can either handle the request directly or pass it to the next middleware component. Middleware can perform various tasks such as authentication, logging, error handling, and more.
   2. **Key concepts**
      1. **Request Delegation**
      2. **Request and Response Handling**
      3. **Asynchronous Processing**
7. **Clean architecture or Onion architecture**
   1. NTT
   2. Use case
   3. Interface adaptor
8. **BAAS Architecture**
   1. Combination of MP and DP2
9. **BIAN Architecture**
10. **Unit Test and Integration test**
11. **Error handling in .net**
    1. Any error which is not handled then middleware will give 500
    2. Middleware is expensive so we are using result pattern
    3. Result pattern – errors are handled
12. **KAFKA architecture. - fulfilment processor, Asynchronous, push notification, inorder to send CD notification**
13. **Github Actions**
14. **Shift left approach**