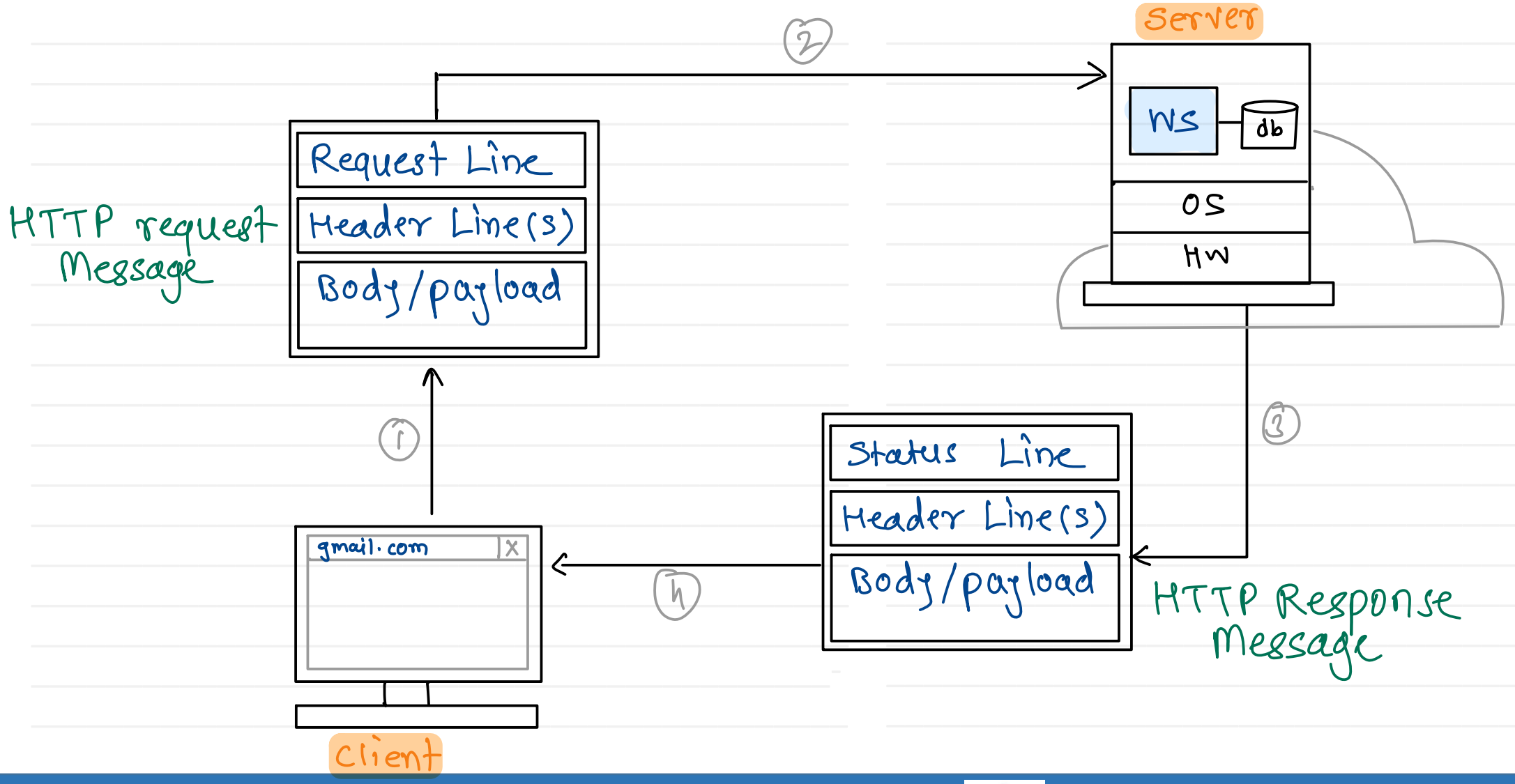
Sunbeam Institute of Information Technology
Pune and Karad

Module – Internet of Things (IoT)

Trainer - Devendra Dhande

Email – devendra.dhande@sunbeaminfo.com

Web Architecture



HTTP

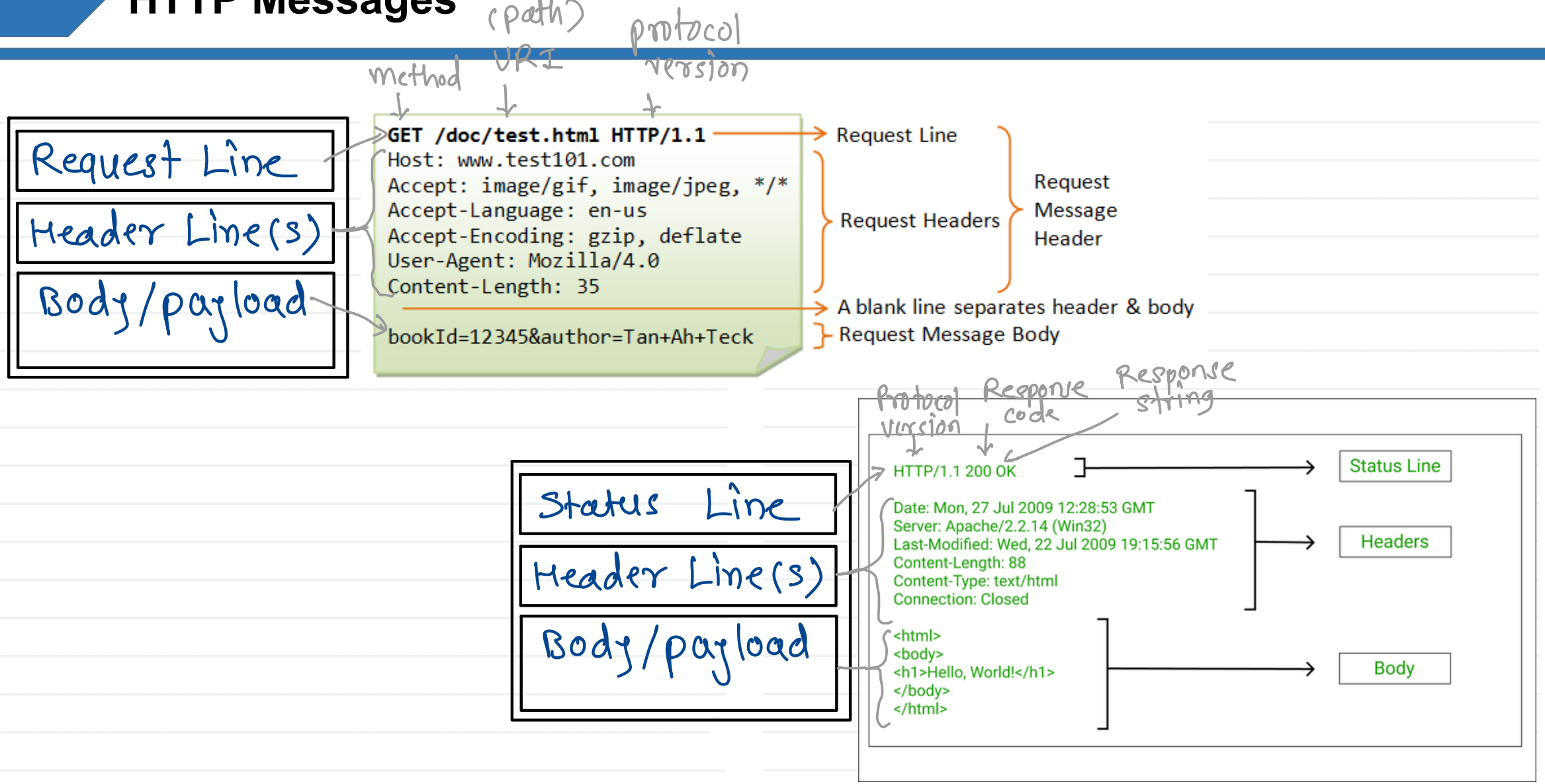
- **HTTP (Hypertext Transfer Protocol)**
- **protocol** of the Internet, enabling the transfer of data between a client and a server
- HTTP is a generic and stateless protocol which can be used for other purposes as well using extensions of its request methods, error codes, and headers.

- **Evolution:**
  - **HTTP/0.9 (1991):**supported only GET method
  - **HTTP/1.0 (1996):**supported many other request methods such as POST, PUT, DELETE, etc
  - **HTTP/1.1 (1997):** offered various new improvements such as pipelining, persistent connections, efficient caching and additional header fields
  - **HTTP/2 (2015):**various performance improvements, like, header compression, multiplexing, and server push which helped in reducing the latency
  - **HTTP/3 (2020):**latest and currently most used version of HTTP. It aims to improve the performance on unreliable networks.

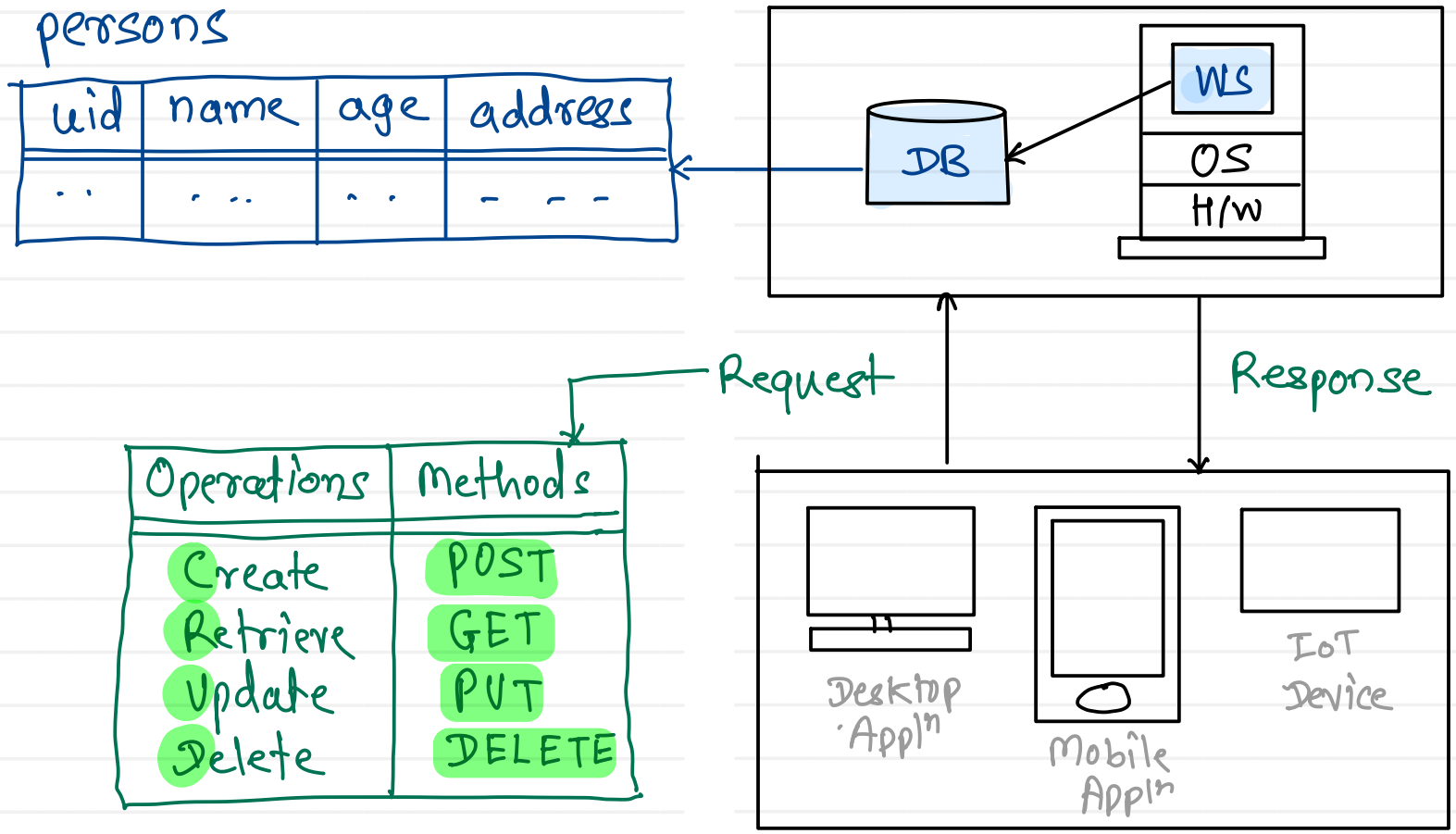
HTTP – Features

- **Stateless:**
  - Each request is independent, and the server doesn't retain previous interaction’s information.
- **Text-Based:**
  - Messages are in plain text, making them readable and debuggable.
- **Client-Server Model:**
  - Follows a client-server architecture for requesting and serving resources.
- **Request-Response:**
  - Operates on a request-response cycle between clients and servers.
- **Request Methods:**
  - Supports various methods like GET, POST, PUT, DELETE for different actions on resources.

HTTP Messages



HTTP methods



HTTP - URL

http://www.google.com

https://www.test.com/login

http://actsfeedback.org/student/login

http://172.18.0.7/exam

http://172.18.0.7:2010/...

1. protocol – http / https
2. server ip / domain name
3. port
4. path of Resource (URI)

HTTP – Status Codes

- **1xx: Informational**
  - the request was received and the process is continuing.
- **2xx: Success**
  - the action was successfully received, understood, and accepted.
- **3xx: Redirection**
  - further action must be taken in order to complete the request.
- **4xx: Client Error**
  - the request contains incorrect syntax or cannot be fulfilled.
- **5xx: Server Error**
  - the server failed to fulfill an apparently valid request.

REST

- REST (**RE**presentational **S**tate **T**ransfer)
- an architectural style for **distributed hypermedia systems**.
- **Roy Fielding** first presented it in 2000
- most widely used approaches for building **web-based APIs (*Application Programming Interfaces*)**
- REST is not a protocol or a standard, it is an architectural style

Principles of REST

- **Uniform Interface**
  - The following four constraints can achieve a uniform REST interface:
    - **Identification of resources** – The interface must uniquely identify each resource involved in the interaction between the client and the server.
    - **Manipulation of resources through representations** – The resources should have uniform representations in the server response. API consumers should use these representations to modify the resource state in the server.
    - **Self-descriptive messages** – Each resource representation should carry enough information to describe how to process the message. It should also provide information on the additional actions that the client can perform on the resource.
    - **Hypermedia as the engine of application state** – The client should have only the initial URI of the application. The client application should dynamically drive all other resources and interactions with the use of hyperlinks.

Principles of REST

- **Client-Server**
  - The client-server design pattern enforces the **separation of concerns**, which helps the client and the server components evolve independently.
  - By separating the user interface concerns (client) from the data storage concerns (server), we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
  - While the client and the server evolve, we have to make sure that the interface/contract between the client and the server does not break.

- **Stateless**
  - Statelessness mandates that each request from the client to the server must contain all of the information necessary to understand and complete the request.
  - The server cannot take advantage of any previously stored context information on the server.
  - For this reason, the client application must entirely keep the session state.

Principles of REST

- **Cacheable**
  - The cacheable constraint requires that a response should implicitly or explicitly label itself as cacheable or non-cacheable.
  - If the response is cacheable, the client application gets the right to reuse the response data later for equivalent requests and a specified period.

- **Layered System**
  - The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior. In a layered system, each component cannot see beyond the immediate layer they are interacting with.
  - A layman's example of a layered system is the *MVC pattern*. The MVC pattern allows for a clear separation of concerns, making it easier to develop, maintain, and scale the application.

- **Code on Demand (*Optional*)**
  - REST also allows client functionality to be extended by downloading and executing code in the form of applets or scripts.
  - The downloaded code simplifies clients by reducing the number of features required to be pre-implemented. Servers can provide part of the features delivered to the client in the form of code, and the client only needs to execute the code.

Thank you!!!

Devendra Dhande

devendra.dhande@sunbeaminfo.com