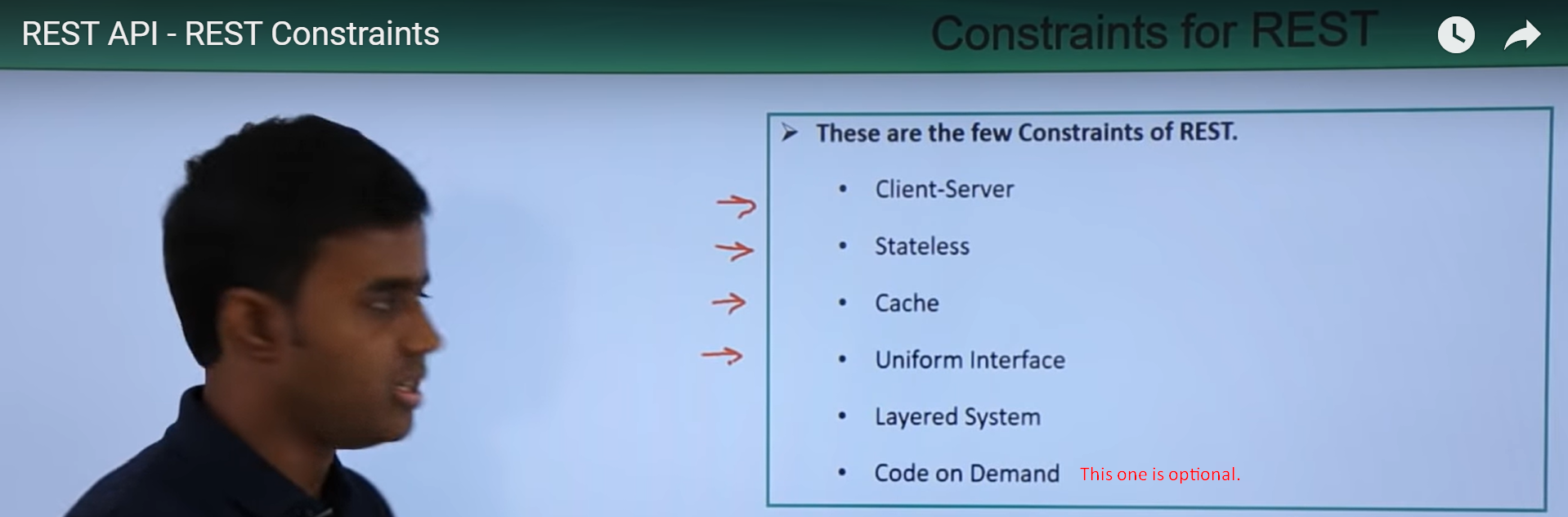
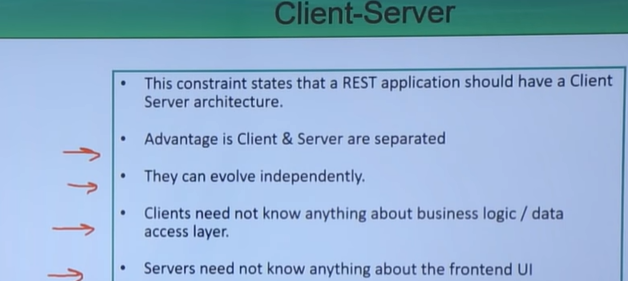
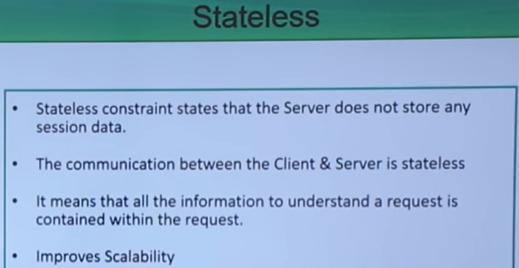
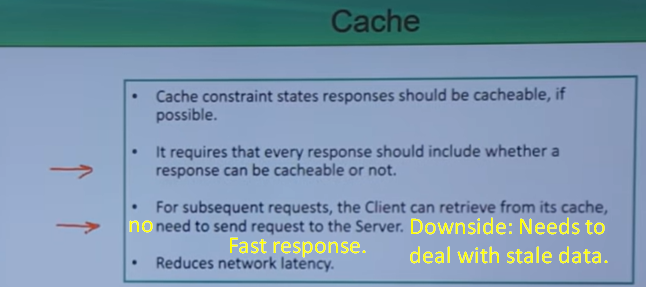
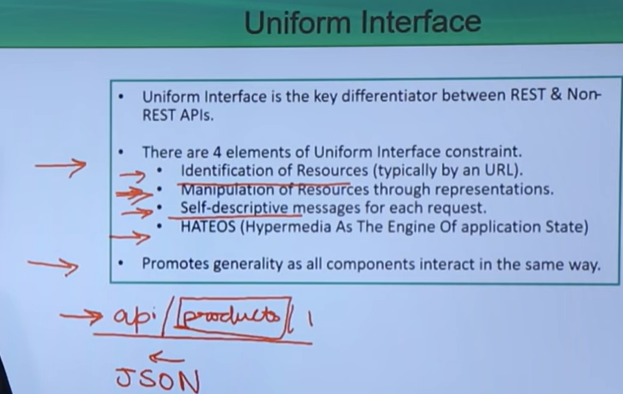
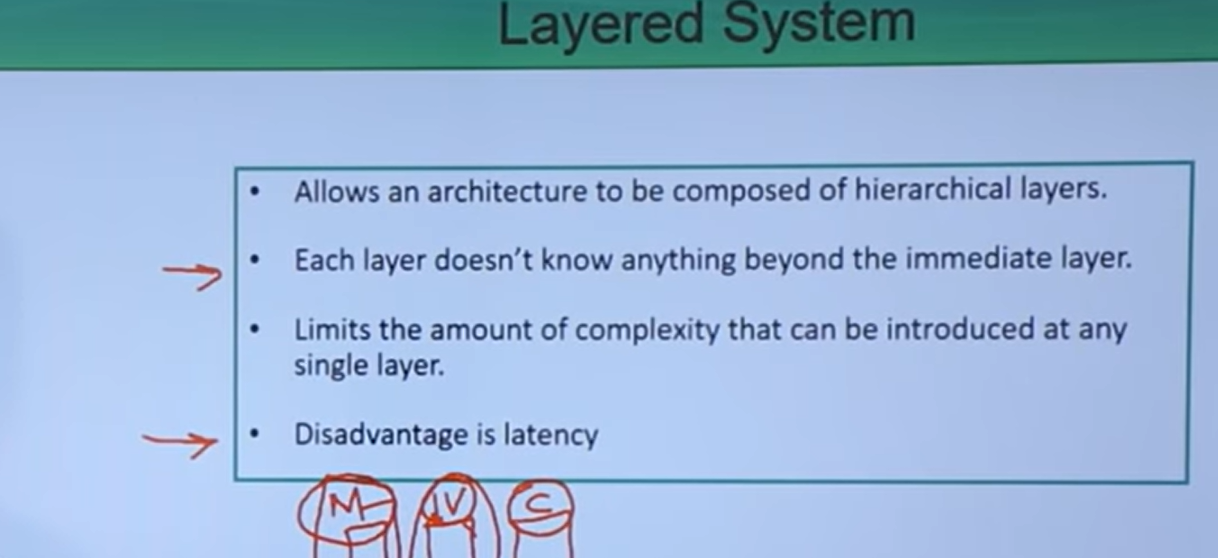
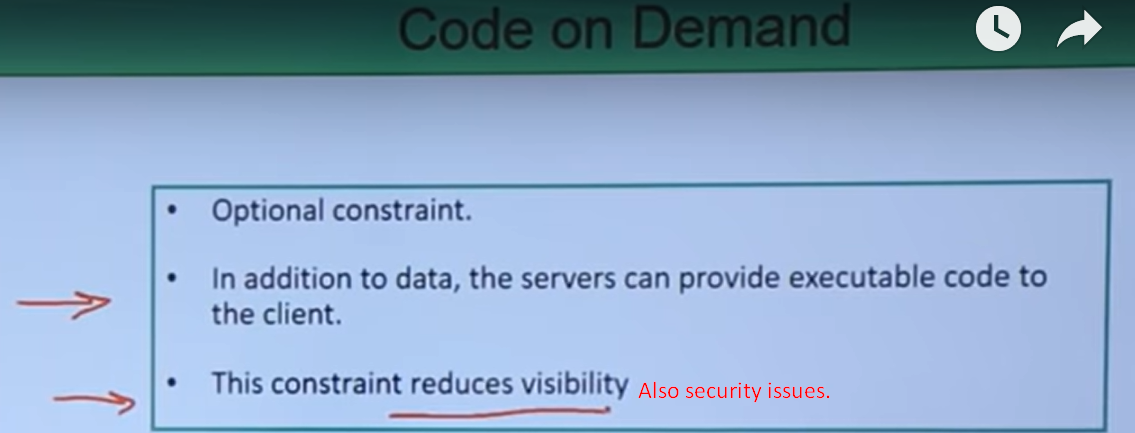
<https://youtu.be/JYNYv8jJQTE> by 

1. 
2. **Client-Server**:  
   
   1. The REST APP should have a client-server architecture.
   2. The client and the server must be separate.
   3. The principle behind this constraint is **the principle of the separation of concerns**.
   4. Same machine can act as both client as well as server.
   5. **Advantages**:
      1. Client and Server are completely separate and because of this, they can evolve independently.  
         Client can move from one version to the other without disturbing the client and same with server.
      2. Client is not worried about business logic and the data needed to process for the request.  
         it is only concerned about hitting the API to get data and process it and display it.
      3. Servers need not to know anything about the frontend UI.
3. **Stateless Constraint**:  
   
   1. Server should not store any session related client data whereas client can store session info in this own context.
   2. Advantage:
      1. Scalable: Easy Scalability.
      2. Visibility: Just looking at the request is enough for server to process the request.
   3. Disadvantage:
      1. More network bandwidth as request has all the meta data.
4. **Cache Constraint**:   
   
   1. When server sends a response, in the response, it should indicate whether the response can be cached by the client and for how long.
   2. Advantage:
      1. It improves the network optimization.
      2. Better performance as response is fast as already cached locally.
   3. Dis-advantage:
      1. Client has to deal with stale data.
5. **Uniform Constraint**:  
   
   1. There are huge varieties of devices our there such as Laptop, PC, Notebook, Mobile, Tablet, Mac.
   2. So, Each and every client device should not require a different way of communicating/interacting with the REST API server.

So, basically these different devices must have uniform way of interacting with the server.

* 1. **There are four elements of Uniform Interface Constraint**.
     1. **Identification of Resources (Typically by an URL)**:
        1. api/**products**/1  
           When we make request like this, basically, we’re identifying resource (product) by id=1
     2. **Manipulation of Resource through representation**:
        1. GET api/**products**/1 : When a client makes this request, he receives the JSON response which is representation for the actual resource on the server but not the resource itself.   
           Now the client must be able to change the resource through this representation.
     3. **Self-Descriptive messages for each request**:
        1. We already discussed in stateless server that the request must be complete in itself with all data and meta data (session info if any).
     4. **HATEOS** (**H**ypermedia **A**s **T**he **E**ngine **O**f application **S**tate)
        1. It means you need to include links for each resource so that the client can discover other resources very easily.
  2. **Advantage**:
     1. It promotes generality as all components interact in the same way.

1. Layered System Constraint:   
   
   1. This constraint tells that the architecture of the system must be composed of multiple layers.
   2. **For example**:
      1. You may have architecture with multiple layers like this where each layer is responsible for a specific function and interacts with the only layer next to it.  
         So, one layer may not know anything about the rest of the layers other than the one immediately next to it.
      2. MVC is best example for this.
         1. M: Deals with data and DB.
         2. V: Deals with output representation to a client.
         3. C: Deals with incoming user requests.
   3. Advantage:
      1. It limits the complexity which can be introduced at any single layer.
      2. Security: If any attack at a particular layer and layers are completely decoupled, that particular security hole is restricted to that layer.   
         So a security breach will never reach the inner architecture.
   4. Dis-advantage:
      1. Latency: A request has to travel through different layers to generate a response.
2. Code on Demand:  
   
   1. In addition to data, sever can send code such as Javascript or Java Applet etc.
   2. **Disadvantage**:
      1. Security: As it deals with the transformation of code so it poses severe Security threats.
      2. **Visibility**: It reduces visibility.