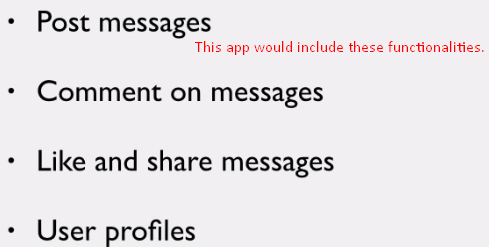
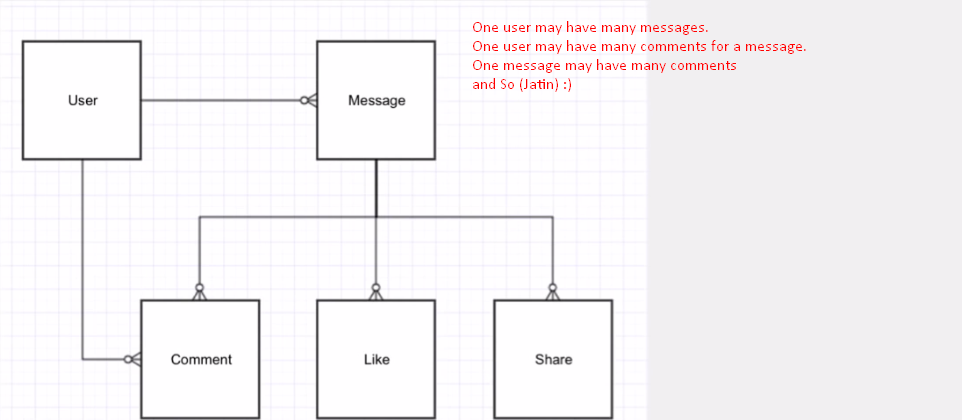
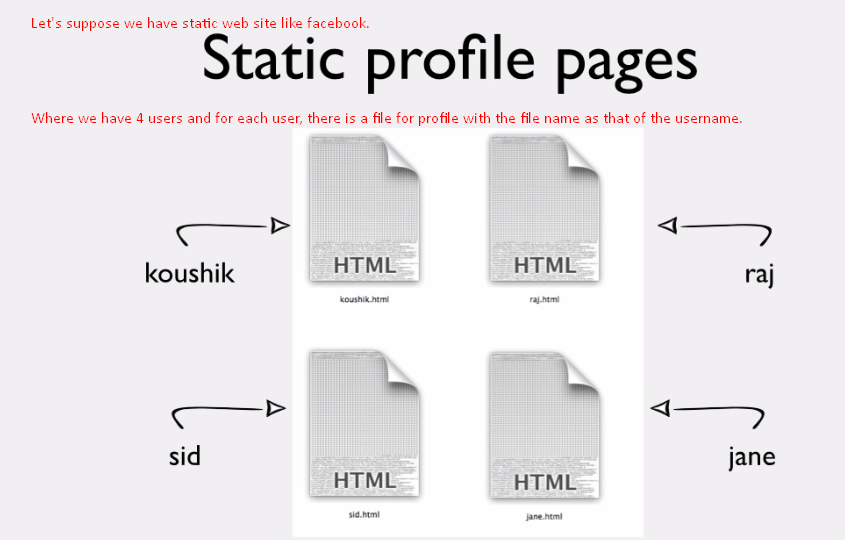
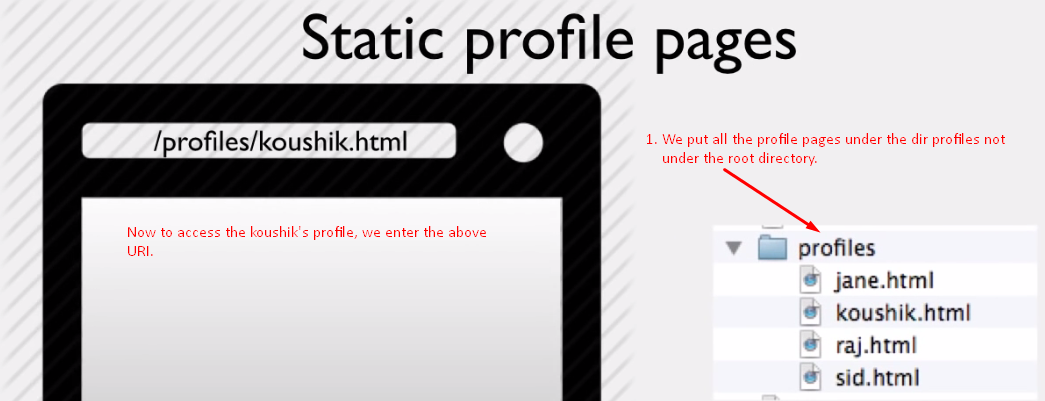
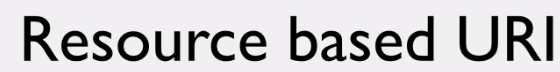
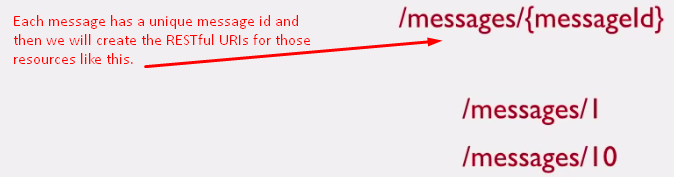
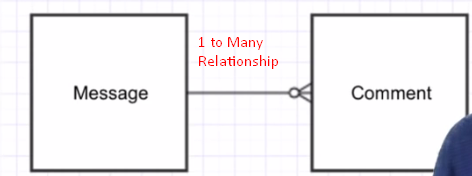
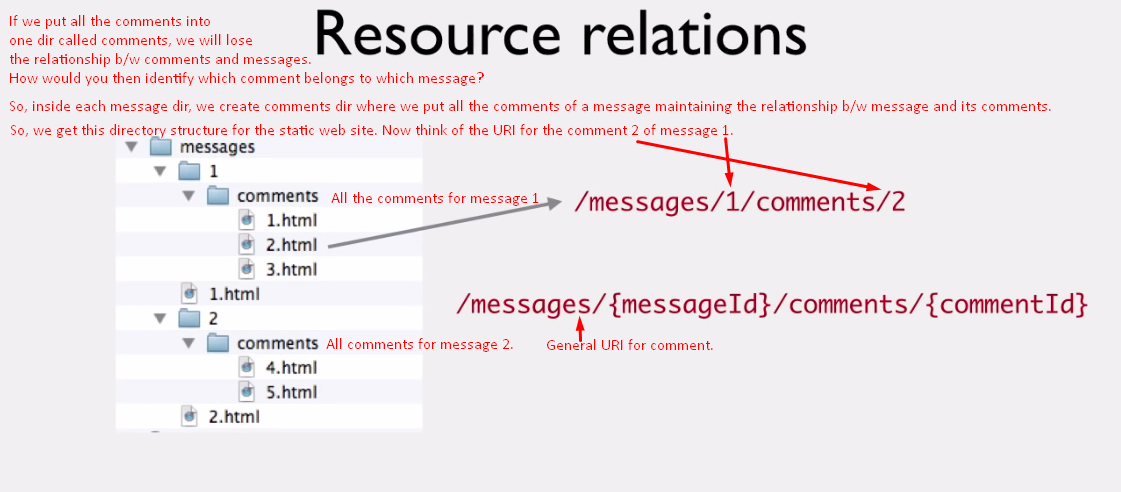
1. 
2. In this and next few lectures, we will see how beautifully to design your RESTful Web Service.
3.   
   
4. 
5.   
   We have to design **RESTful Resourced Based URI**.
6. First, suppose that it is web app not RESTful app, then what would be URI for the request to get a message page based on message id?
7. Actually, its design depends on the framework, you’re using. Suppose, you’re using structs.
8. 
9. In the following slide, there is a mistake about “human which needs to remember” which must be “needs not to remember”.
10. But in case of URIs in the RESTful WS App, the programmers need to remember the URIs to make the HTTP calls to these URIs.   
    So, there needs a **common URIs convention** for different entities (**resources**). So that they don’t need to struggle to find out what URI is for what.   
    This is where RESTful concept of URI coming into picture.
11. 
12. Every page has a unique URL that is unique and standard.   
    **NOTE**: When designing RESTful URI, think of static pages. 
13. 
14. 
15. 
16.   
    That is your first “**Resourced Based RESTful URI**”.  
    You’ve to think of resources and create unique URIs for them.
17. Let’s take other examples. As on messenger app, we’ve the ability to post a message and what would be URI for it.
18. 
19. Notice two things in these all URIs.
20. URIs contains nouns not verbs.
    1. /**messages**/{messageId}
    2. /**profiles**/{profileName}
    3. **NOTE**: You don’t have verbs like 🡺 /getMessages/{messageId}
    4. Noun is name for the resource itself. Being static web site, you would not create dir something like “getProfiles” but rather dir 🡺 profiles
21. Secondly, resource name is plural 🡺 **messages**, **profiles**; they are not singular as it makes very clear that there are multiple messages, profiles.
22. This is the first step to designing a RESTful API for any system. You’ve to identify the **entities** and they become the **resources**.
23. The **main advantage** of this Resource Based URIs is that they **don’t depend on any frameworks** so there is no “.do or .action” and **no query parameters** in your URIs as these details are not significant to the RESTful Clients.  
    In future, if you switch to another framework, the designed URIs doesn’t get affected.
24. Let’s look at other nouns in our messenger application such as comments, likes, and shares🡸 all the resources.
25. Take comment id 20 and what is the best RESTful URI for that based on the model we discussed?
26. 
27. 
28. When designing URI for resources, you may encounter with resources dependent on other resources.  
    Take the example of message and comment. One message has multiple comments and each comment has a comment id. 1 to many relationship.    
    It doesn’t acknowledge the relationship b/w message and comment resource.
29. 
30. 

This URI makes it very clear that a particular comment belongs to a particular message. So, the relationship b/w message and comment is well-established.  
To know comment, you need to know 2 things. First the message id and second the comment it.  
It’s totally up to the service provider to design either **/messages/{msgId}/comments/{commId} or /comments/{commId}.** If you want the consumer must have the message id to access the comment, go for the first one otherwise second one in case comment id is enough. 

1. 
2. **Summary**:  
   🡺 Identity resources.  
   🡺 Design Resource Based URIs.  
   Now, you may be wondering why messages are not related to profiles? Why messages are first level entities? As messages are posted in a particular profile so there is one-to-many relationship b/w profile and message. So, couldn’t you have message URI like this.  
   
3. You could. But we have decided to have message as separate entity. Again, the design all depends on you.