in-class problem APIs

Morgan Bergen

EECS 368 Programming Language Paradigms

Fri Oct 21 16:28:51 CDT 2022

considering the API between the client browser and the filer server that we talked about in previous lectures and that you are writing for Assignment 5:

1. Is the API an RPC? Explain why.

An RPC is an protcol for an API, an API is not an RPC. An RPC (remote procedural call) is a type of protocol for APIs. In cloud computing/distributed computing, a remote procedure call is when a program causes a procedure to execute in a different address space. This differing address space is typically another computer on a shared network, which is coded as if it were a normal call.

2. Is the API SOAP? Explain why.

What soap is to an api is the same as what an rpc is to an api. Thus SOAP is a major type of protocol for APIs. SOAP stands for simple object access protocol, it is within the messaging protocol type family and it's protocol is for the specification for exchanging structed information in the implementation of web services in computer networks. SOAL uses XML information set for its messaging format and relied on aapplication layer protocols. Mostly it needs hypertext transfer protocol, as well as SMTP simpel mail transfer protocol for messaging and transmission.

SOAP allows developers to process invokcations running on differing operating systems in order to autheticate, authorize, and communcate using xml. SOAL allows clients to invoke web services as well and recieve responses independent of the language or platform.

3. Does the API follow the REST client-server architecture style? Explain why.

REST APIs in modern web applications typically have six guiding constraints for architectural styles such as client-server architecture.

This form of styling is often said to be RESTful. This form of design enforces something called separation of concerns which means that it separates the user interfaces from the data storge, it allows for improved scalability, and allows components to evolve independently, which is necessary in an internet-scale enviroments.

4. Does the API follow the REST stateless archiectural style?
   Explain why?

Yes, statelessness means that every HTTP request happens in complete isolation. When the client makes an HTTP request, it includes all info necessary for the server to fulfill the request and it does not rely on information from previous requests from the client, which is a pinicle aspect when the client sends a request. This increases performace by removing server load caused by retention of session information and is ideal for high volume applications.

5. Does the API follow the REST cacheability architectural style? Explain why.

Yes, the cacheability architectural style on the world wide web can cache responses for clients and intermedaries. Responses must define themselves in order to prevent clients from providing data that may be integral to the user. This allows for improved scalability and performaces.

6. Does the API follow the REST layered system architectural style? Explain why.

Yes, a client cannot ordinarily tell whether it is connected directly to the end server or not. If a proxy is placed between the client and server it won't affect their communication and there would need to be any updates for the client or the server code. Intermediary servers can improve system scalability by enabling load balancing and providing shared caches. Intermediary servers can call multiple different other serveres to generate a client reponse.

7. Does the API follow the optional REST code on demand archiectural style? Explain why.

Yes, rest code on demand can temporarily have servers extend the functionality of their client by transferring executable code.

6. Is it a RESTful API? Explain why.

A RESTful API is an interface that two computer systems use to exchange information securely over the internet, technically there really isnt a difference between the two other its constrainst, REST is the set of constraits and RESTful refers to an API adhering to thiose constraints, it can be used for a variety of different architecture systems including web services, web applications, operating systems, and general software.