Caleb Chang
CS355 - A5

APIs (Application Programming Interface) define how a user can interact with a system via network requests. This allows users to access a systems database, processing power, and utilities. There are many styles which APIs follow. Here are three styles, how they differ, how they are similar, and when one would use one style over the other.

The RPC pattern for APIs takes every function on a system, and converts it to an endpoint. This means the functions do a very specific task and send back a very specific response. This means the network payload is small, fast, and high-performing. This API pattern is also very straightforward. A function is defined to do something, if you call a function it will do what it's defined to do, and that’s it. Although this pattern is straightforward, as the quantity of functions grows, it can be difficult to grasp what the API does. In other words, this pattern does not offer any abstraction between the system and the client.

The REST API pattern focuses on resources. Endpoints specify resources instead of functions. This pattern provides a high level of abstraction between the client and the server. This pattern also returns a lot of metadata. If you go to the root of the API, the root will tell you what the API does. There will be links to other resources, and if you follow those links, the API will respond with metadata about what that particular resource contains, and what it does. Although this pattern specifies to focus on resources, there is no agreement on how to implement this. There are structures like HAL, JSON-API, and Ion which provide a structure, but there is no standard. REST APIs telling you how to navigate the API with each call can be both beneficial and detrimental. This is good because the user can navigate the API more easily. This is detrimental because payloads are big, or network performance is low. Furthermore, if you are looking for a specific detail about a specific resource, you will get a lot of information you don’t need.

GraphQL APIs differ from both REST and RPC APIs by the method of communication. Whereas RPC and REST use endpoints, GraphQL uses queries. A schema defines the types of queries a GraphQL API may perform, and these queries can be chained. Consequently, GraphQL can retrieve very specific, nested data in one call whereas a REST API may need many network communications to do this. So GraphQL network performance is high(good). GraphQL is very good at representing graph-like, or database-like resources. GraphQL can be disadvantageous because it is more complex than the others. GraphQL does not implement versioning.

These three API patterns each have their advantages and disadvantages, and different situations will be best suited for different patterns. A small application with few functions should use an RPC pattern due to its simplicity. An application focused more on actions and not on commands

should use RPC. An application which focuses on resources should use REST or GraphQL. If these resources have a great number of relationships with other resources, therefore resulting in a database like structure, GraphQL is best. Otherwise, REST is better due to its relative simplicity. If network performance is a concern, GraphQL or RPC is better than REST.

Although RPC, REST, and GraphQL API styles may be perceived as different, there are actually many similarities between these styles, and it can sometimes be difficult to identify an API in one category. For example, the “difference” between REST and RPC APIs is that REST focuses on resources while RPC focuses on functions. However, one can think of each endpoint in a RESTful API as a function which returns a resource, but functions mean RPC. And in the talk by Nate Barbettini, Nate says an RPC API is like taking all the functions of a system, and turning it into an API. Therefore, RPC APIs are less discoverable and tightly coupled. Considering the system is written well, and with selective omission of functions which perform specific tasks, converting a system’s functions to an API can result in an API with high discoverability and low coupling. At this point, one may call this API RESTful, but it is still focused on functions. Furthermore, Nate talks about how REST APIs can return a lot of information and therefore have network problems, in reality, RESTful APIs allow users to specify queries which can filter results. In this sense, RESTful APIs are very much like GraphQL APIs. An actual API is not defined by a particular style. An actual API is defined by the tasks it performs, and it can use just one style, all three styles, styles not mentioned in this video, or unconventional styles not yet seen.