A Review of Data-Driven Architecture with RESTful APIs

Gabriel Ajabahian

A Review of Data-Driven Architecture with Restful APIs

In computer programming processes, Application Programming Interfaces (APIs) refer to sets of tools, protocols and routines from which software applications are built. The APIs express software components in terms of output, input, underlying type, and operations. Therefore, APIs aid in defining functionalities independent of particular representations. This process allows the implementations and definitions to vary (while ensuring that interfaces are not compromised). Indeed, good APIs make it easier for individuals to develop programs by offering the required building blocks, upon which programmers put the blocks together (Haupt et al., 2014, 3). On the other hand, Representational State Transfer (REST) constitutes a style through which software architecture operates on distributed systems. One of the distributed systems is the World Wide Web in which the REST design model has emerged predominantly (Perez et al., 2011, 283).

RESTful conforms to the REST constraints. Six constraints characterized the system. The constraints include client-server, stateless, cacheable, layered system, uniform interface, and code on demand. The client-server constitutes a uniform interface responsible for separating clients from servers. The separation implies that clients do not engage in data storage events. Rather, the data storage process operates internally with each server; improving the client code.

Stateless forms a client-server communication in which client context stored on servers is not constrained further between requests. Given that each client request contains all the necessary data, session states are held in the client (Perez et al., 2011, 284). Cacheable enables the clients to cache responses. The responses may explicitly or implicitly define themselves as cacheable or not. The definition prevents clients from re-using inappropriate or stale data while responding to additional requests (Haupt et al., 2014, 3).

Layered systems serve to improve system scalability by providing shared caches and, enhancing load-balancing. In addition, layered systems enforce security policies because clients cannot ordinarily define whether they are connected to intermediaries along the way or, the end users. The code on demand, an optional application, enables the servers to temporarily customize or extend client functionality by transferring executable codes. Lastly, uniform interfaces operate between servers and clients to decouple and simplify the architecture (Haupt et al., 2014, 4). Therefore, constraint compliance precedes conformation to REST architectural styles. The conformation enables distributed hypermedia systems to exhibit desirable emergent qualities such as reliability, portability, visibility, modifiability, simplicity, scalability and performance.

References

Haupt, F., Karastoyanova, D., Leymann, F. & Schroth, B. 2014. *A model-driven approach for REST compliant services*. IEEE Computer Society: University of Stuttgart, Germany

Perez, S., Durao, F., Melia, S. Dolog, P. and Diaz, O. 2011. *RESTful, Resource-Oriented Architectures: A Model-Driven Approach*. Springer-Verlag Berlin Heidelberg