The API Design Process

Published on May 30, 2018

Bernard Stibbe
Senior ICT-Architect bij VZVZ

18 articles + Follow

A good service begins with a good definition of resources. Resources are entities which are exposed to service consumers (clients). Clients need to be able to identify these resources over a network and obtain access to them. For this purpose, resources need to be given an ID, which in REST is the URL (Uniform Resource Locator).

Zooming into the term – **resources**, we quickly find out that these are not just any entities, but rather *business entities* (entities defined in terms of a business domain). Therefore grouping them under business domains is only natural. The business domain defines the boundaries, the

rules and the eco-system / bounded context under which these resources live. The way chosen to represent these resources is through models, called the "**REST Domain Model**".

A REST Domain Model is a collection of resources. A resource is a view or a projection on a business object which is to be exposed to clients. It can be modeled as an *UML Class* with stereotype <<*resource*>>, in order to differentiate from other objects (which are not resources). All resources are placed in package with a domain name of the domain where the resource definition is owned. A resource has a set of properties of a specific data type. Resources are modeled using the object oriented principles of encapsulation and abstraction. In other words, the resources contain properties which belong to them in a functional sense. They should not contain properties of other objects. This is typically seen in situations where one wishes to add identifiers of other objects directly as properties to the

object instead of modeling an association between them. Although this might simplify the object structure, the functional meaning is lost and should be avoided.

A resource is rarely, a stand-alone entity. It often has relationships with other entities and resources. A REST Domain Model is, therefore, a collection of resources and the relationships between them. Each relationship has a cardinality or multiplicity between other entities. These relationships can be broadly categorized into *Inheritance* and *Association*.

An API (**Application Programming Interface**) represents a contract between a provider and zero or more consumers for accessing resources via a service. Because API consumption is a programmatic exercise, it is important to get a clear picture of what kind of business entities offers the API and how those API resources should be approached.

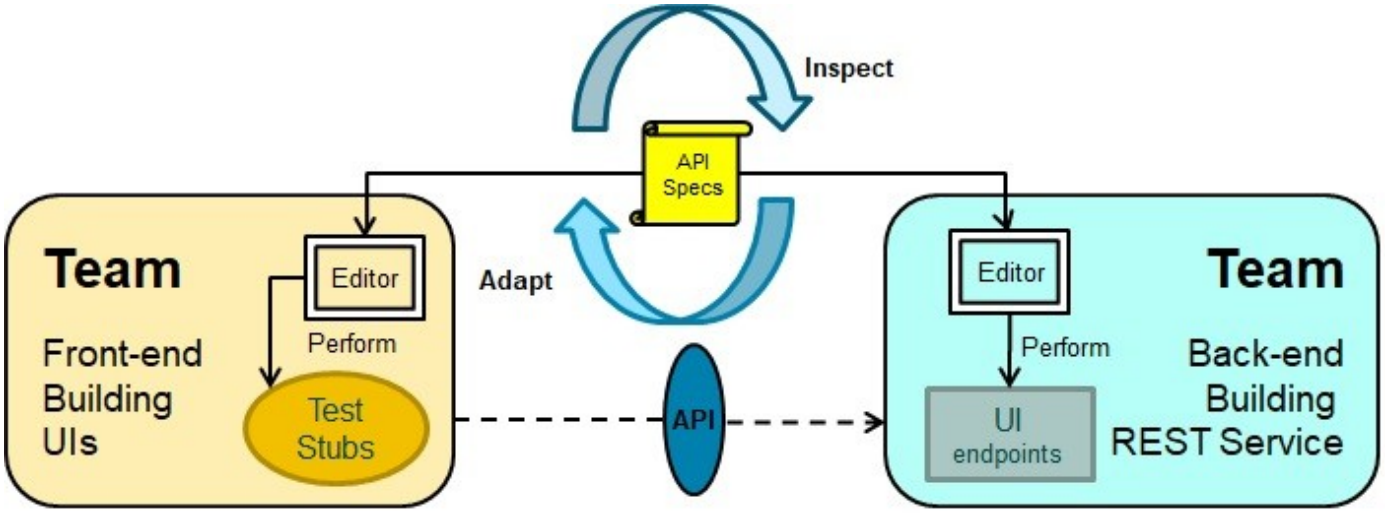
This is called the "**REST Service Contract or Interface Specification**". It helps application developers to assess and use the capabilities offered to see if the options meet their needs before developing their own (front- or backend) systems based on the specification.

In the Interface Specification you will see, the definition of resources, schema's, operations, parameters, status codes, security rquirements and representation examples of the API.

When API development is outsourced, the interface specification can be written in abstraction as a means to define the API that the supplier must build. The specification can be treated as a separate entity, is version-controllable and it should be possible to use it as the main API documentation or reference. Various API modeling languages are available for defining those interface specification. Some are patented (eg RAML,

API Blueprint) while others are more technical (eg WADL).

One of the most common today's API modeling language is the "*OpenAPI specification*" (Swagger) that is managed by the Open API Initiative. This initiative is supported by many IT companies. It offers a standard, programming-language-agnostic (JSON / YAML) interface for REST APIs with which one can discover and understand the possibilities of the service offered. It is important to use a modeling language to define the specification of the API interface, because this is one of the important parts of the API product and can be stored and managed.



Start with :

- REST Domain Model
- Define Interface Specification from REST Domain Model
- Write (Swagger) Specification
- Store (Swagger) Specification in Repository
- Generate API stubs (JAX-RS, Node.js, Spring MVC, etc)
- Implement (conceptual) API