2023-04-09

Rethinking API and Web

Abstract

We live in an era when RESTful is nothing new and CGI is considered obsolete. APIs are always something over HTTP, and are often designed in the RESTful way. But changes are happening and news ideas like GraphQL are emerging.

It is time to decouple APIs from HTTP.

Businesses Are Not RESTful

The real world business operations are not RESTful.

How can we abstract the operation of approving some applications into resource operation? There is no "approve" verb in HTTP. We may use POST to create a "ticket for approval", but that is against the very idea of RESTful.

And what response code should we expect for a batch operation when some approvals are successful and others are not? We cannot receive an array of 200 and 403.

Although sometimes useful and great, RESTful is a poor choice for abstraction and generalization. In the real-world life, we work by sending messages in the OOP way (SmallTalk); messages are often class method invocations (POST in RESTful, INSERT in SQL) and instance method invocations (GET/PUT/DELETE in RESTful, SELECT/UPDATE/DELETE in SQL).

Businesses Are Not Databases

GraphQL is the other extreme. It appears to be claiming "let frontend developers CRUD for themselves and forget API endpoints".

It is almost-always true that all backend system data will be stored by the database, especially in an era where the "cloud-native" movement is popular (along with all the YAML nonsense).

Some systems require plugins in order to implement logging, audition, alerting, message bus, microservice hotplugging, etc. They may be seen as side effects in the eyes of database IO. The database IO pattern hides such complexities from the eyes of frontend developers, in a scary way. For example, in the account creation

scenario, sending the verification email is a side effect. The database IO pattern creates an illusion for the frontend developer that simply creating a user entity in the database is enough, where the side effect of sending a verification email is hidden. It is possible that the email address verification daemon works as an observer for the database IO, but I believe that calling "myapp.useradd(...)" is a more graceful way.

Also, the database IO pattern exposes some other unnecessary complexities. It requires a frontend developer to know too much about the database table structure. While one may argue that this way does not create extra data structure knowledge beyond the universal data model documentation in a given project, I still believe that complex operations should be left for the backend. Take the account creation scenario for example. The frontend developers should not care too much about how Person (uid, email address, username, etc) and Shadow (uid, hashed password, salt, etc) are organized in the database, nor should the frontend care too much about how the token is generated for the verification email. Also notice that the web frontend may not be the only consumer of APIs; other consumers may exist (e.g. third parties).

It is possible that some HTTP daemon works as an router for GraphQL, who decides how certain read requests may be served by something other than the underlying MariaDB connection. However, again, database IO is a poor choice for API abstraction.

Businesses Are Commands

Unlike real method invocations (Java), remote API calls do not have real references (RAM pointers). Therefore, basic RPC pattern is not enough. Remote APIs are basically RPCs, but not exactly RPCs. Instead, instances must be identified by some ID. And this has something in common with the RESTful pattern.

Suppose that HTTP was never invented and every user had to do their operations over SSH. A user connects to the server and types "send-friend-request bob" to send a friend request to Bob and types "show-news-feed" to get his latest news feed.

CLI is a great abstraction for universal API wrapping. It can happen locally or remotely. It natively fits the request-response pattern. Its underlying remote communication foundation can be SSH, WebSocket, and HTTP. Commands and responses are exchanged in an authenticated secure session which fits the HTTP header authentication token (cookie, etc) infrastructure.

Instead of "POST /api/friendReqest" (with form body "target=bob"), we can request "POST /api/cmd" with a JSON body like the following code block, where the field name "argm" means "argument map".

```
{
  "cmd": "send-friend-request",
  "argm": { "target": "bob" }
```

```
}
```

Also, for batch operation...

```
POST /api/batch-cmd
Content-Type: JSON
Auth-Token: 1145141919810
[
  {
    "cmd": "send-friend-request",
    "argm": { "target": "bob" }
  },
  {
    "cmd": "send-friend-request",
    "argm": { "target": "david" }
  }
]
```

A more radical statement may be: businesses are shell scripts. And the backend developer should provide relative commands so that these commands may be invoked by the user via web frontend or mobile app or SSH or any other channel.