API Architecture and Design

REST and all the rest

PTS3 / FMFI UK / 3.4.2025

Michal Kostič

Outline

1. Motivation
    a. Concept of API
    b. HTTP based APIs
2. REST Theory
    a. Definition
    b. Principles
    c. Richardson Maturity Model
3. REST in Practice
    a. Designing REST API step-by-step
4. GraphQL

Motivation

Need for common interface

- Example - e-commerce site
  - Customer buys goods
  - Company employees manage the inventory and orders
- Multiple client applications
  - Browser single page application
  - Android and iOS apps
  - Desktop app for back office
  - External customer integration
  - Automation scripts (e.g. AI orders items missing from the fridge)
- Goals
  - Share logic
  - Share common state (inventory)
- Solution
  - All clients talk to server via common interface called **Application Programming Interface - API**

What is an API?

- Application Programming Interface (API) is a mechanism that allows two software components to communicate
- Defining Features
  - Expose functionality via stable contract
  - Hide implementation
- Benefits
  - Functionality reuse
  - Increased developer efficiency
  - Potential for unexpected innovation
- Mind the humans!
  - **API is the user interface for developers**

Example: Service API Technologies

Service is a software component that provides functionality accessible over network through well defined interface (e.g. payment service)

- XML-RPC
  - https://www.tutorialspoint.com/xml-rpc/xml_rpc_request.htm
- SOAP
  - https://developers.google.com/ad-manager/api/soap_xml
- gRPC (protocol buffers)
  - https://www.oreilly.com/library/view/grpc-up-and/9781492058328/ch04.html
- GraphQL
  - https://www.apollographql.com/blog/making-graphql-requests-using-http-methods
- REST
- Real-life

HTTP APIs

- A type of API that uses HTTP protocol for communication
- Advantages
  - Ready infrastructure, open ports
  - Browser support
  - Developer friendly
  - Human readable
- Disadvantages
  - Chatty HTTP protocol and inefficient character encoding - compare custom binary APIs
  - Low support for state - compare message queue with replayability
- Examples
  - RPC style API
  - SOAP
  - REST
  - GraphQL

Example: Programming language vs action APIs vs REST

Code

```
// get item
shoppingCart.items["1234"];
// add item
shoppingCart.items().add(item);
// increase quantity
shoppingCart.items[item.id].quantity++;
```

Action Based

```
// get item
POST /shopping-cart/get-items?cartId=1234
// add item
POST /shopping-cart/add-item?cartId=1234
// increase quantity
POST /shopping-cart/update-item?cartId=1234&quantity=2
```

REST

```
// get item
GET /shopping-cart/1234/items
// add item
POST /shopping-cart/1234/items
// increase quantity
POST /shopping-cart/1234/items
```

REST API Theory

What is Representational State Transfer (REST) API?

Any HTTP API that transfers JSON

API that performs actions via transfer of the representation of the state

An API that conforms to constraints defined by the REST architectural style

What is Representational State Transfer (REST) API?

Any HTTP API that transfers JSON ❌

API that performs actions via transfer of the representation of the state

An API that conforms to constraints defined by the REST architectural style

What is Representational State Transfer (REST) API?

Any HTTP API that transfers JSON ❌

API that performs actions via transfer of the representation of the state ±

An API that conforms to constraints defined by the REST architectural style

What is Representational State Transfer (REST) API?

Any HTTP API that transfers JSON ❌

API that performs actions via transfer of the representation of the state ±

An API that conforms to constraints defined by the REST architectural style ✔

REST API

- REST was formally defined in dissertation [Fielding, 2000], but since then got into heavy popular usage
- Formally defined as
  - Representational State Transfer (REST) [is an] architectural style for distributed hypermedia systems, describing the software engineering principles [...] and the interaction constraints chosen to retain those principles[...]. REST is a hybrid style derived from several of the network-based architectural styles [..] and combined with additional constraints that define a uniform connector interface.
- REST API is API that conforms to the REST architectural style

More info:
https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

REST Architectural Principles

- Client-Server decoupling
- Statelessness
- Cacheability
- Uniform interface - resources, URIs
  - All systems share the same interface (HTTP)
  - The business logic is in the payload (messages/state representation)
  - Interface constraint
    - Identification of resource
    - Manipulation of resources through representations
    - Self-descriptive messages
    - Hypermedia as the engine of application state
- Layered system architecture
- Code on demand (optional)

Richardson Maturity Model

- How RESTful is the API?
- Level 0 - Swamp of POX (Plain Old XML)
  - RPC style calls
- Level 1 - Resources
  - RPC towards individual resources
- Level 2 - HTTP Verbs ⇐ Borderline REST API
  - HTTP request methods instead of actions
- Level 3 - HATEOAS ⇐ True REST
  - Links to other Hypertext As The Engine Of Application State

More info:
https://martinfowler.com/articles/richardsonMaturityModel.html

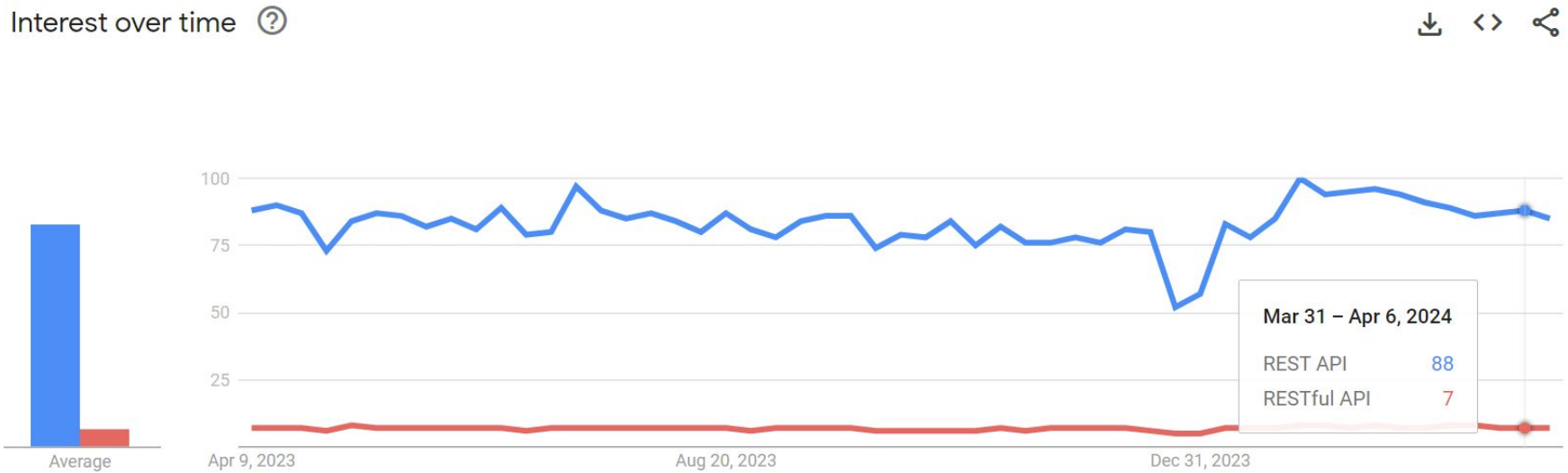
Data Elements

- Resource
  - Anything worth referencing by itself
  - E.g. document, physical object, temporal service (“weather in Bratislava today”)
- Resource identifier
  - URI = Universal Resource Identifier
  - https://example.com/api/v1/shopping-cart/1234/item/789
- Representation
  - JSON, XML, HTML
- Metadata
  - Representation - media-type
  - Resource - source link, alternates
  - Control data - cache-control, ETag

Terminology: REST API vs RESTful API

REST - name of the architectural style

RESTful - adhering to the REST architectural style



REST in Practice

Building REST API

- Understand your domain, use cases and requirements
  - Think for a moment if REST API is best choice or other type of API is better
  - Model API closely to domain
- Identify resources
  - Nouns, objects or process info
- Identify URIs
  - URI unique per resource
  - Primary keys in the URI, additional parameters in the query parameters
  - For strong ownership relation - consider sub-resource (can get inflexible)
- Model the resource representations
  - Follow domain objects as closely as practical
  - Links vs inline objects (HATEOAS)
  - Absolute links vs relative links
- Map actions to HTTP request methods
  - GET
  - POST vs PUT

Understand your domain

Understand your domain
Identify resources
Identify URIs
Model the resource representations
Map actions to HTTP request methods

- Best APIs closely model the domain
- Consider appropriateness of the REST API
- Example:
  - As part of the e-commerce site, we want to provide users with ability to manage items in their shopping cart. Users can add, remove and change number of items in their shopping cart.
  - Each item is identified by an internal identifier, contains a text, unit and unit price.
  - Once the user finishes shopping, they create an order from the shopping cart item.
  - The order is then delivered to a customer address.

Understand your domain
Identify resources
Identify URIs
Model the resource representations
Map actions to HTTP request methods

Identify Resources

- Review your requirements
- Identify nouns, objects or process info
- Example:
  - Shopping cart
  - Shopping cart item
  - Order
- Common pitfall - How to handle processes and actions?
  - Object representing the process status
  - Use action based URI (not strictly REST, but pragmatic)

Identify URIs

Understand your domain
Identify resources
Identify URIs
Model the resource representations
Map actions to HTTP request methods

- Each resource needs URI
- Primary keys in the URI, additional parameters in the query parameters
- For strong ownership relation - consider sub-resource (can get inflexible)
- Singular vs plural
- Examples
  - ```
    /shopping-carts/1234
    /items/7890
    /items?shopping-cart-id=1234&item-id=7890
    /shopping-carts/1234/items/7890
    ```
    ```

Model Resource Representation

Understand your domain
Identify resources
Identify URIs
Model the resource representations
Map actions to HTTP request methods

- Best APIs closely model the domain
- Follow domain objects as closely as practical
- Links vs inline objects (HATEOAS)
  - Absolute links vs relative links
- Common pitfall - How to represent monetary values and other decimals?
  - String
  - Minor units (e.g. cents)
- Example:

```
{
    "id": 1234,
    "items": [
        {
        "id": 7890,
        "name": "Apple",
        "unit": "pieces",
        "amount": "10",
        "unit_price": "2.07"
    }],
    "_links": [
        {"rel": "self", "href": "https://www.example.com/api/v1/shopping-carts/1234"},
        {"rel": "customer", "href": "https://www.example.com/api/v1/customers/5678"}
    ]
}
```

Map Actions

Understand your domain
Identify resources
Identify URIs
Model the resource representations
Map actions to HTTP request methods

- Use HTTP request methods
  - GET - get selected representation of the resource - does not alter state of server (safe)
  - POST - request resource to process the entity based on resource's rules (NOT idempotent)
  - PUT - replace the resource with entity
  - DELETE - delete the specified resource
  - PATCH - apply partial modifications to a resource
- Examples
  -

```
GET /shopping-carts/1234/items/7890
POST /shopping-carts/1234/items/
PUT /shopping-carts/1234/items/7890
```

Versioning

- How to indicate version
  - URI
    - /api/v1/shopping-carts/1234
  - Header (not obvious which version)
    - Accept-version: 1
  - Query parameter
    - /api/shopping-carts/1234?version=1
  - Port
    - https://example.com:8001/api/shopping-carts/1234
  - Media type
    - Accept: application/vnd.xm.device+json; version=1
- When to version - backwards compatibility
  - Add required request parameter
  - Remove path
  - Renaming required attribute - not backwards compatible if not keeping original attributes

Error handling

- Goal is to communicate internet with the API integrator (consumer/client)
- Less is more
  - Success
    - 200 OK
  - Client error
    - 400 Bad Request
  - Server error
    - 500 Internal Server Error
- Standard errors
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not found
  - …
- My favorite additions
  - 422 Unprocessable Content - the request is well formed, but does not make sense
  - 502 Bad Gateway - failure due to 3rd party request

Authentication

- Shared secret
  - Auth-secret-key: 12357938745983459
- JWT
  - Authorization: Bearer 12345678…
- OAuth
  - Authorization: Bearer 897978…

Documentation

- OpenAPI specification
  - Contract first - generate code from specification
  - Code first - generate specification from code
- Live documentation website - Swagger
  - Examples
    - https://petstore.swagger.io/#/
    - https://editor.swagger.io/
- Developer portal
  - Quickstart
  - Tutorials
  - Examples
    - https://docs.stripe.com/development
    - https://www.twilio.com/docs/conversations
- Client libraries
  - Make integration easier by API wrappers in popular programming languages

GraphQL

Motivational Example

- We need to list item name, quantity and producer name in the shopping cart
  - REST API - multiple calls (particularly if opting for HATEOAS)
  - OR use more denormalized schema, however it can start growing
- Better solution?

GraphQL

- Query language and server runtime
- Flow
  - Define a schema for the objects (server)
  - Send queries (client)
  - Queries are populated by resolvers (server)
  - For updates define mutations (essentially actions on server)

GraphQL Example

```
type Query {
    shoppingCart(cartId: String): ShoppingCart
}
```

```
query CondensedItems {
  shoppingCart(cartId: "123") {
    items {
      product {
        name
        producer {
          country
        }
      }
      quantity
    }
  }
}
```

```
type ShoppingCart {
    customer: Customer
    items: [ShoppingCartItem]
}

type ShoppingCartItem {
    product: Product
    quantity: Int
}

type Product {
    name: String
    producer: Producer
}

type Producer {
    name: String
    country: String
}
```

```
shoppingCart(obj, args, context, info) {
    return {
        "customer": {
            name: "John Doe"
        },
        items: [
            {
                product: {
                    name: "Apple",
                    producer: {
                        name: "Apple Corporation",
                        country: "USA"
                    }
                },
                quantity: 7
            }
        ]
    }
},
```

Q&A

https://www.michalkostic.com
https://linkedin.com/in/michalkostic
michal.kostic@gmail.com