CS 1520: Recitation 10

RESTful Services

What is an API?

- Application Programming Interface

- “An application program interface (API) is code that allows two software programs to communicate with each other”.

- GUI vs API
  - GUI provides interface to connect user to a program while API connects a program to another program.
  - GUI -> HTML, CSS etc. API -> XML, JSON

https://searchmicroservices.techtarget.com/definition/application-program-interface-API

Example

- Hypothetical Example
  - The waiter is an API that takes request (order) from the client (customer) and brings back response (food) from the server (kitchen)

- Real Life Example
  - Different websites allowing users to post Facebook comments to their article
  - Websites that show tweets as part of their article
  - All these websites use the Facebook/Twitter API for this purpose

https://www.youtube.com/watch?v=s7wmiS2mSXY

REST

- Representational State Transfer

- “It is a software architectural style that defines a set of constraints to be used for creating web services”

https://en.wikipedia.org/wiki/Representational_state_transfer

Architectural Constraints of REST

- Client-Server architecture
  - Client and server application must be able to evolve separately and independently

- Stateless
  - All client-server interaction should be stateless

Architectural Constraints of REST

- Layered System
  - A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way
  - Improves system scalability by enabling load balancing and by providing shared caches
  - Also enforces security policies
  - Example: Deploy APIs in Server A, store date in server B and authenticate requests in server C

Architectural Constraints of REST

- Cacheable
  - Responses must implicitly or explicitly, define themselves as cacheable or not to prevent clients from getting stale or inappropriate data in response to further requests
  - Reduces client-server interaction
  - Improves scalability and performance

- Code-on-Demand (optional)
  - Servers can temporarily extend or customize the functionality of a client by transferring executable code

Architectural Constraints of REST

- Uniform Interface
  - Resource Identification in requests
    - Using URIs
    - Nouns not Verbs
  - Resource Manipulation through representations
    - A client holding a representation of a resource, has enough information to modify or delete the resource
  - Self-Descriptive messages
    - Each message includes enough information to describe how to process the message
  - HATEOAS

REST Resource Naming Convention

- A resource can be:
  - Collection
    - Example URI: /customers
  - Singleton
    - Example URI: /customers/{customerId}
  - Sub-collection resource
    - Example URI: /customers/{customerId}/accounts

REST Resource Naming Convention

- RESTful URI should refer to a resource that is a thing (noun) instead of referring to an action (verb)

- Resource archetypes can be divided into four categories:
  - Document
  - Collection
  - Store
  - Controller

Naming based on resource archetype

- Document
  - Singleton resource inside a collection
  - Use *singular* name
  - Examples
    - http://api.example.com/user-management/users/{id}
    - http://api.example.com/user-management/users/admin

Naming based on resource archetype

- Collection
  - Server-managed directory of resources
  - Use *plural* name
  - Example:
    - http://api.example.com/user-management/users
- Store
  - Client managed resource repository
  - Use *plural* name
  - Examples:
    - http://api.example.com/cart-management/users/{id}/carts
    - http://api.example.com/song-management/users/{id}/playlists

Naming based on resource archetype

- Controller
  - A controller resource models a procedural concept
  - Controller resources are like executable functions
  - Use verb
  - Example:
    - http://api.example.com/cart-management/users/{id}/cart/checkout

HATEOAS

- Hypermedia As The Engine Of Application State
  - Component of REST application architecture
- A client interacts with a network application whose application servers provide information dynamically through hypermedia
- No documentation needed to use a website
  - Example: Home page of a website provides link to other resources in the website

Example

- Consider a JSON data of messages that you get as response to “www.abc.com/messages”

```
[
        {
                “id”: “1”,
                “message”: “Hello world”
                “author”: “James”
        },
        {
                “id”: “2”,
                “message”: “Hello universe”
                “author”: “Harry”
        }

]
```

Example

- In order to access the details of message with “id” = 1 from the server, the user needs to hardcode the following URI in the browser

www.abc.com/messages/”1”

Example

- To avoid this, the response from the server can contain a link to message with “id” = “1”

```
[
        {
                “id”: “1”,
                “message”: “Hello world”
                “author”: “James”
                “href”: “www.abc.com/messages/1”
        },
]
```

Example

- The response can also have links to other resources on the web server

```
[
        {
                "id": "1",
                "message": "Hello world"
                "author": "James"
                "href": "www.abc.com/messages/1"
                "comments-href": www.abc.com/messages/1/comments"
                "likes-href" : "www.abc.com/messages/1/likes"
        },
]
```

Example

- However, it is cumbersome to remember the keys names in the returned data

- Solution: Use the “rel” attribute
  - Gives the relation between the current document and the linked doument

```
[
        {
                "id": "1",
                "message": "Hello world"
                "author": "James"
                "links": [
                        {
                                "href": "www.abc.com/messages/1"
                                "rel": "self"
                        }
                        {
                                "href": www.abc.com/messages/1/comments"
                                "rel": "comments"
                        }
                        {
                                "href": "www.abc.com/messages/1/likes"
                                "rel": "likes"
                        }
                ]
        },
]
```

References

- https://restfulapi.net/rest-architectural-constraints/
- https://en.wikipedia.org/wiki/Representational_state_transfer
- https://restfulapi.net/resource-naming/
- https://spring.io/understanding/HATEOAS
- https://www.youtube.com/watch?v=NK3HNEwDXUk