PART 2. LARGE SCALE DATA STORAGE SYSTEMS
DATA EXCHANGE MODEL

Sangmi Lee Pallickara
Computer Science, Colorado State University
http://www.cs.colostate.edu/~cs435

FAQs

• Term project presentation
  • 10 minutes per team
  • Presentation
  • Q&A
  • Transition
• Submit your slides 2 hrs before the class starts via Canvas

Topics

• Data Exchange Model
  • RESTful service interface

Part 2. Large scale data storage system
Data Exchange Model

Wearable devices and sensors
Fitbit APIs

- Store, read, analyze user’s activity data
- Data collected from user’s devices are stored in anywhere available
- Immediate and historical analysis

For more information: https://dev.fitbit.com/build/reference/

Example: Activity & Exercise Logs

GET https://api.fitbit.com/1/user/[user-id]/activities/date/[date].json

user-id: The encoded ID of the user. Use "-" (dash) for current logged-in user.
date: The date in the format yyyy-MM-dd
Accept-Language: The measurement unit system to use for response values.
Accept-Language: The locale to use for response values.

Example: Activity & Exercise Logs: Response

```json

"activities": [
  {
    "activityId": 51007,
    "activityParentId": 90019,
    "calories": 230,
    "description": "7mph",
    "distance": 2.04,
    "duration": 1097053,
    "hasStartTime": true,
    "isFavorite": true,
    "logId": 1154701,
    "name": "Treadmill, 0% Incline",
    "startTime": "00:25",
    "steps": 3783
  }
],
```

Example: Activity & Exercise Logs: Response

```
"goals": {
  "caloriesOut": 2826,
  "distance": 8.05,
  "floors": 150,
  "steps": 10000
},
```

Example: Activity & Exercise Logs: Response

```
"summary": {
  "activityCalories": 230,
  "caloriesBMR": 1913,
  "caloriesOut": 2143,
  ...
```

Who are providing REST interfaces?

- Google Cloud Storage Service
- Google Search REST
- Netflix
- Twitter
- Flickr
- Amazon eCommerce
- Amazon S3
- ...
Part 2. Large scale data storage system

Data Exchange Model

RESTful Service

Representational State Transfer (REST)

• An architectural style for networked hypermedia applications
• Used to build Web services that are lightweight, maintainable and scalable

• RESTful service
  • A service based on REST

• REST is not dependent on any protocol
  • But, almost every RESTful service uses HTTP as its underlying protocol

To be a REST client

• Endpoint

```java
.header("X-Mashape-Key", "gaNm1SHvGmLoY2AZU8kLH&Rp1zET7ckjshnQGZ1lf9w")
.header("Accept", "text/plain")
.asString();
```

This material is built based on,


RESTful services

• REST is NOT a standard

• It uses components that are based on standards
  • HTTP
  • URL
  • XML/HTML/GIF/JPEG/etc (Resource Representation)
  • Text/xml, text/html, image/gif, etc (MIME Types)

Results (Using Java)
4 major HTTP methods for REST CRUD

- POST – Update
- GET – Read
- PUT – Create
- DELETE – Delete

When to use GET

- Caches depend on the ability to serve cached representations
- Without contacting the original server
- Safe and idempotent information retrieval

Methods can also have the property of "idempotence" in that (aside from error or expiration issues) the side-effects of N > 0 identical requests is the same as that for a single request.

When to use POST

- If it is not safe to cache
- Make the response noncacheable
- Add a Cache-Control: no-cache header
- Consider any possible side effects
- Implement servers which can handle frequently repeatable operations (e.g. concurrent access)
When to use **POST**

- To create a new resource (sub-resource)
- To run a query with large inputs
- To perform any unsafe or non-idempotent operation (when no other HTTP method is available)

**Continued**

- Originally, POST was designed for
  - Annotation of existing resources
  - Posting on group articles
  - Creates a child resource
  - Providing append operations for database
  - E.g. Create a resource that lives under /items resource

- Unsafe and non-idempotent processing for the server

**Creating Resources Using POST**

- Submit a POST request with a representation of the resource to be created by the factory resource
- Optional Slug header
  - Name of the new resource **suggested by clients**

**POST request**

```
# Request
POST /user/smith HTTP/1.1
Host: www.example.org
Content-Type: application/xml;charset=UTF-8
Slug: Home Address

<address>
  <street>1, Main Street</street>
  <city>Some City</city>
</address>
```

**POST Response**

```
# Response
HTTP/1.1 201 Created
Location: http://www.example.org/user/smith/address/home_address

<address>
  <id>urn:example:user:smith:address:1</id>
  <adtom:link rel="self" href="http://www.example.org/user/smith/address/home_address"/>
  <street>1, Main Street</street>
  <city>Some City</city>
</address>
```

---

**Part 2. Large scale data storage system**

**Data Exchange Model**

RESTful Service: **PUT**
Creating Resources Using **PUT**

- **PUT** requests that the enclosed entity be stored under the supplied URI
- **PUT** is idempotent
- Use **PUT** to create/add new resources only when clients can decide URIs of resources
  - Otherwise, use **POST**

In RFC of HTTP,

The fundamental difference between the **POST** and **PUT** requests is reflected in the different meaning of the Request-URI. The URI in a **POST** request identifies the resource that will handle the enclosed entity. That resource might be a data-accepting process, a gateway to some other protocol, or a separate entity that accepts annotations. In contrast, the URI in a **PUT** request identifies the entity enclosed with the request -- the user agent knows what URI is intended and the server MUST NOT attempt to apply the request to some other resource. If the server desires that the request be applied to a different URI, it MUST send a 301 (Moved Permanently) response; the user agent MAY then make its own decision regarding whether or not to redirect the request.

---

Is **PUT** idempotent?

- Is **DELETE** idempotent?
  - Yes

PUT request

```plaintext
# Request
PUT /user/smith/address/home_address HTTP/1.1
Host: www.example.org
Content-Type: application/xml; charset=UTF-8
Slug: Home Address

<address>
  <street>1, Main Street</street>
  <city>Some City</city>
</address>
```

What if there are two conflicting **PUTs**?

- HTTP/REST does not require "lock" for these concurrent access.
- REST is **STATELESS**.
PUT Response

# Response
HTTP/1.1 201 Created
Location: http://www.example.org/user/smith/address/home_address
Content-Type: application/xml;charset=UTF-8

<address>
  <id>urn:example:user:smith:address:1</id>
  <atom:link rel="self" href="http://www.example.org/user/smith/address/home_address">
    <atom:id>urn:example:user:smith:address:1</atom:id>
  </atom:link>
  <street>1, Main Street</street>
  <city>Some City</city>
</address>

POST example

# A SOAP message tunneled over HTTP POST
POST /Messages HTTP/1.1
HOST: www.example.org
Content-Type: application/soap+xml; charset=UTF-8

<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
               xmlns:enc="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:ns="http://www.example.org/messages">
    <ns:DeleteMessage>
      <ns:MessageId>1234</ns:MessageId>
    </ns:DeleteMessage>
  </soap:Body>
</soap:Envelope>

DELETE response

# Using DELETE
DELETE /message/1234 HTTP/1.1
Host: www.example.org

HTTP/1.1 202 Accepted
Content-Type: application/xml;charset=UTF-8

<status xmlns:atom="http://www.w3.org/2005/Atom">
  <status>pending</status>
  <atom:link rel="self" href="http://www.example.org/task/1">
    <atom:id>urn:example:task:1</atom:id>
  </atom:link>
  <message xml:lang="en">Your request has been accepted for processing.
</message>
<created>2009-07-05T03:10:00Z</created>
<ping-after>2009-07-05T03:15:00Z</ping-after>
</status>

Is this a good design?
Part 2. Large scale data storage system

Data Exchange Model

RESTful Service: Managing Errors

How to Return Errors

- Error needs to be represented as well
- Errors in the clients’ input
  - 4xx status code
- Error due to server implementation or current state
  - 5xx status code
- Include a Date header
  - The date-time at which the error occurred

Description of Error

- Formatted and localized document (HTML or plain text) included in a body
  - Except for the HEAD method
- Other details can be linked via a Link header or in the body
- Keep the body descriptive

Error Message

```xml
<error>
  <message> Account limit exceeded. </message>
</error>
```

Include your error code in the Header

- 400 Bad request
- 401 unauthorized
- 403 forbidden
- 404 not found
- 409 conflict
- 410 gone
- 412 precondition failed
- 413 request entity too large
- 415 unsupported media type
Include your error code in the Header

- 500 Internal Server Error
- 503 Service Unavailable

Provide description

- A brief message describing the error condition
- A longer description with information on how to fix, if applicable
- An identifier for the error
- A link to learn more about the error condition, with tips on how to resolve it

Example of a Good Error message

```xml
<?xml version="1.0" encoding="UTF-8"?>
<error xml:lang="en" xml:base="http://www.example.org"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <message>Account limit exceeded. We cannot complete the transfer due to insufficient funds in your accounts</message>
  <error-id>321-553-495</error-id>
  <account-from>urn:example:account:1234</account-from>
  <account-to>urn:example:account:5678</account-to>
  <atom:link href="http://example.org/account/1234" rel="http://example.org/rels/transfer/from/>
  <atom:link href="http://example.org/account/5678" rel="http://example.org/rels/transfer/to/>
</error>
```