Western Norway
University of
Applied Sciences

DAT152 – Advanced Web Applications

Web Services I

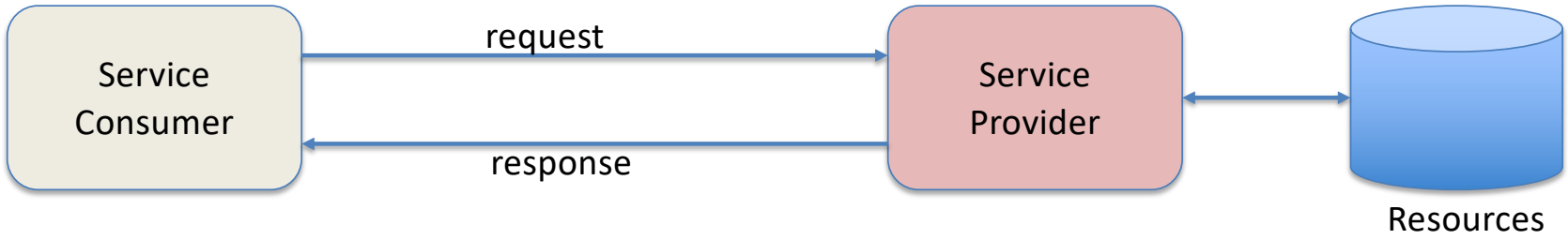
Today

- Background/History - Web Services
  - (RPC, SOAP, WSDL, UDDI, etc.)
- RESTful Web Services

Web Service?

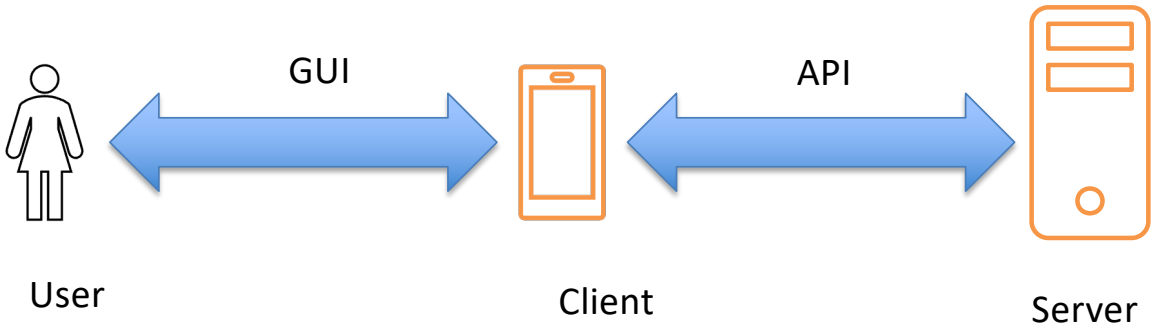
https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice

- A web service is a software system designed to support **interoperable** ***machine-to-machine*** interaction over a network…
- A Web service is a set of related application functions that can be programmatically invoked over the Internet.



Web Services API

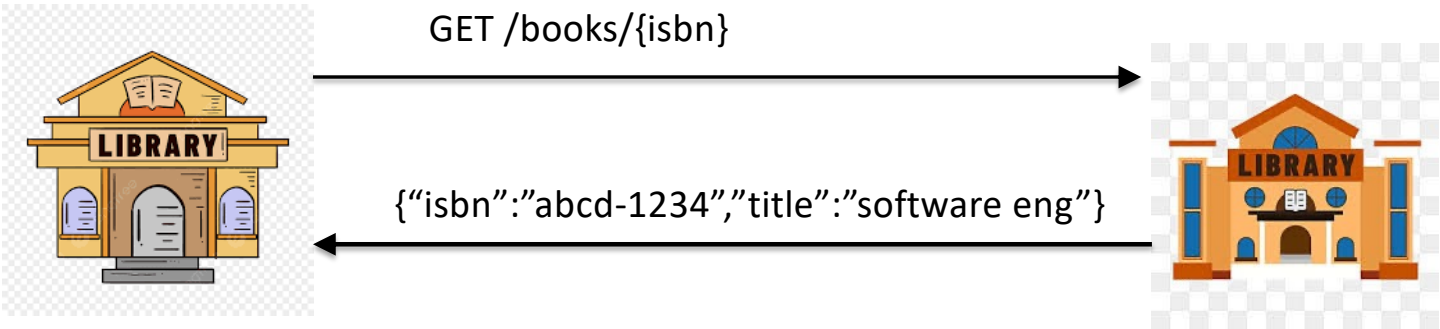
- A GUI provides interface for Human <-> Machine communication



- An API is an interface for Machine <-> Machine communication

Some Examples

- Amazon (e-Commerce)
- StackOverflow (Q&A Forum)
- Interbank transactions (e.g., transfer, balance,)
- Weather reports
- Travel advisories
- X (Messaging)
- Library services (e.g., remote book reservation)



Web Services API Design

- API design will consider (among others):
  - Method – e.g. retrieve, delete, modify, compare
  - Scope/Parameters – e.g. goods in a category
  - Data – e.g. info about an item
  - Other parameters – e.g. sorting field, filtering, pagination
  - Return value: e.g. list of goods
  - Error message: What type of error, where, and hint?

Web Services API Design

- Web APIs should facilitate interactions between a service consumer and a service provider
- An application can provide services such as:
  - Get all books
  - Update book list
  - Delete book
  - Add new book
  - Borrow a book
- A service can bind to specific method
  - e.g., ‘Get all books’ -> public List<Book> getBooks();

Web Services API Design

- RPC-based
- RESTful

RPC-based approach

- RPC = Remote Procedure Call
- Based on method call
- Service-oriented since the service/method are the central elements

RPC-based approach

For a complex call – XML-RPC

```
POST /xmlrpc HTTP 1.0
User-Agent: myXMLRPCClient/1.0
Host: 192.168.124.2
Content-Type: text/xml
Content-Length: 169
<?xml version="1.0"?>
<methodCall>
   <methodName>circleArea</methodName>
      <params>
         <param>
            <value><double>2.41</double></value>
         </param>
      </params>
</methodCall>
```

Each business application can provide own service API based on own specification. However…

Interoperability?
Discoverability?

RPC-based approach

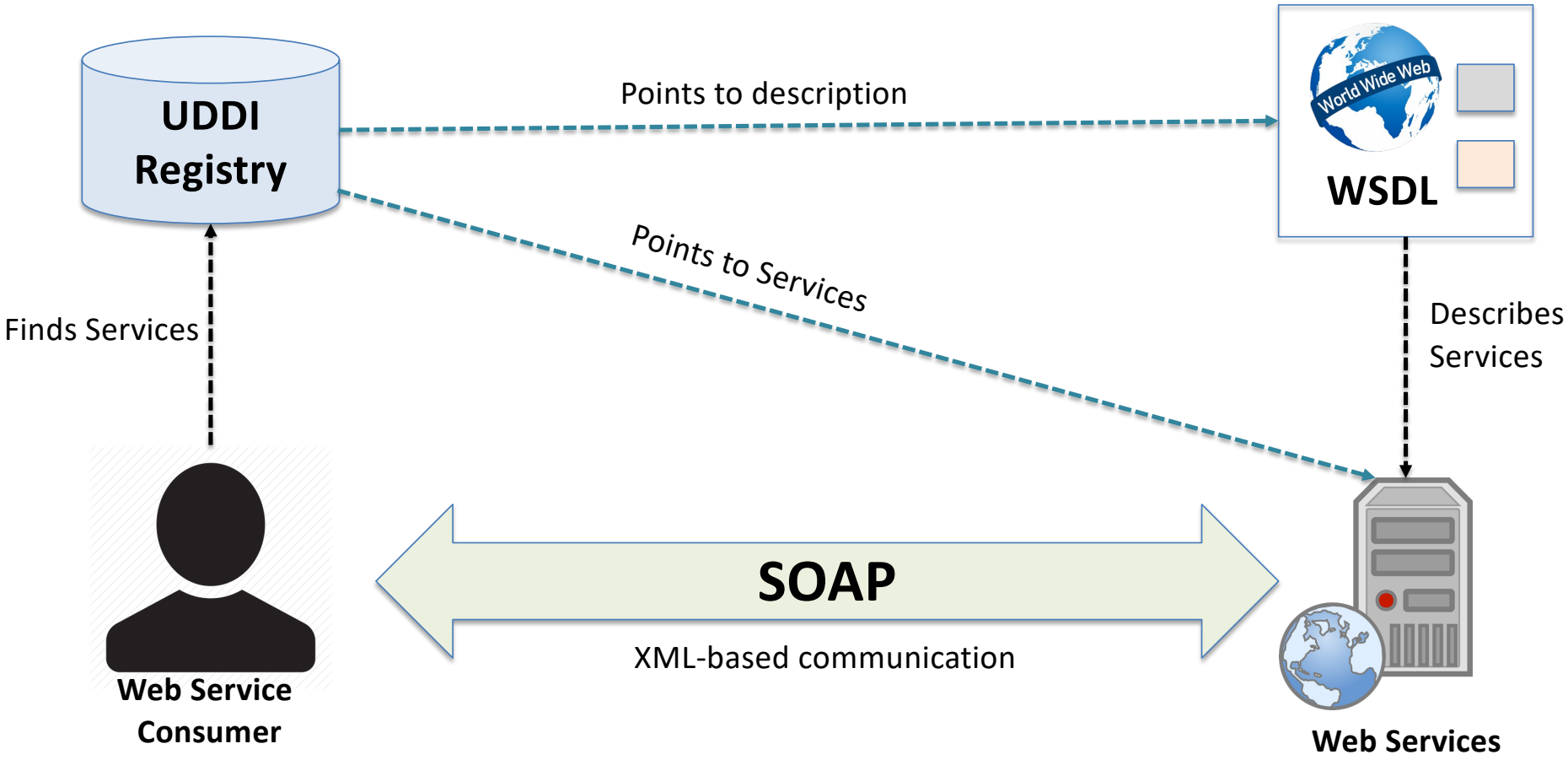
- Web Services Standards
  - A set of standards for web services to exchange information over the internet
  - Preventing individual vendors from imposing own standard on the internet

RPC-based approach

W3C recommendations (standards) for developing Web APIs for Web Services

- SOAP (Simple Object Access Protocol)
  - XML-based **protocol** for exchanging information over internet protocols (e.g. HTTP, FTP, etc.)
- WSDL (Web Services Description Language)
  - XML-based specifications describing interfaces to and instances of web services on the network
    - Endpoints, Operations, Request Structure, and Response Structure
- UDDI (Universal Description, Discovery and Integration)
  - A global business registry

RPC-based


UDDI
Registry
Points to description
World Wide Web
WSDL
Points to Services
Finds Services
Describes
Services
SOAP
XML-based communication
Web Service
Consumer
Web Services

SOAP Request - Example

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:gs="http://spring.io/guides/gs-producing-web-service">
   <soapenv:Header/>
   <soapenv:Body>
      <gs:getCountryRequest>
         <gs:name>Spain</gs:name>
      </gs:getCountryRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Method

Scope

SOAP Response - Example

```xml
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header/>
    <SOAP-ENV:Body>
        <ns2:getCountryResponse xmlns:ns2="http://spring.io/guides/gs-
producing-web-service">
            <ns2:country>
                <ns2:name>Spain</ns2:name>
                <ns2:population>46704314</ns2:population>
                <ns2:capital>Madrid</ns2:capital>
                <ns2:currency>EUR</ns2:currency>
            </ns2:country>
        </ns2:getCountryResponse>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Critics – SOAP Standards

- ‘The problem is, most of today’s “web services” have nothing to do with the Web. In opposition to the Web’s simplicity, they espouse a heavyweight architecture for distributed object access’.
- Today’s “web service” architectures reinvent or ignore every feature that makes the Web successful (RESTful Web Services by Leonard Richardson & Sam Ruby).
- From Service-oriented architecture (**SOA**) to resource-oriented architecture (**ROA**)

REST as alternative to SOAP

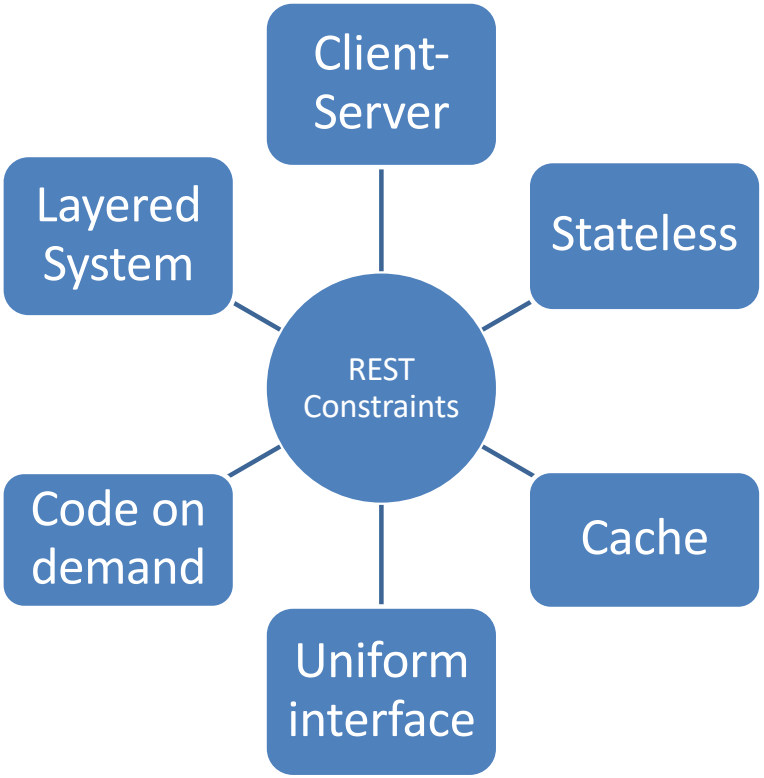
- The question is, can we simply leverage the Web HTTP APIs to deliver resources
- Roy Fielding proposed REST architectural style in his PhD thesis (2000)
  - REST (**R**epresentational **S**tate **T**ransfer)
  - It is an architectural style for designing loosely coupled applications over the network, often for development of web services.
  - It is not a standard
  - does not enforce any rule regarding how it should be implemented at the lower level, it just put high-level design guidelines and leaves us to think of our own implementation.

REST in a Nutshell

- Services offer **resources**
- All resources have a unique **URI**
  - **URIs** tell a client there is a resource somewhere
  - **Clients** can request a specific representation of the resource as allowed by the server
- HTTP **verbs** are used to retrieve or manipulate resources in a universal way

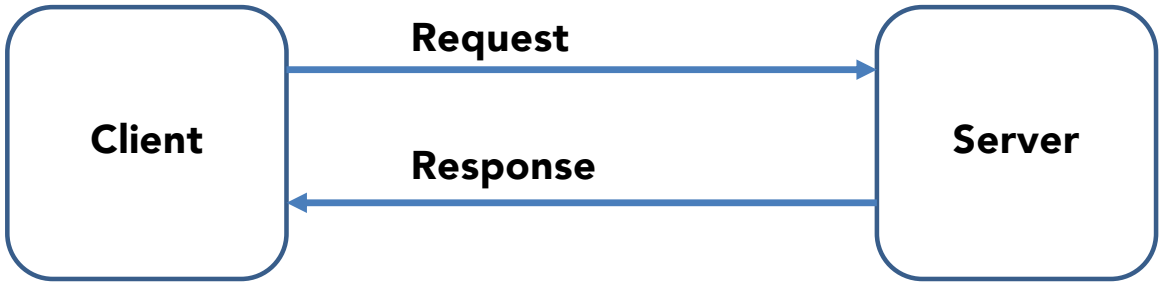
REST Architectural Constraints

REST defines 6 architectural constraints to make any web service a truly RESTful API



Client-Server

- Employs client-server architectural style
  - Separation of concerns constraints
  - Client applications and server applications must be able to evolve separately without any dependency on each other



Stateless

- This constraint dictates that each request from client to server must contain all of the information necessary for the server to understand the request
  - The server will not store anything about the latest HTTP request the client made
  - It will treat every request as new
  - No session, no history on the server
  - Session state is kept entirely on the client

Cache

- Adds cache constraints to form the client-cache-stateless-server style.
- It requires server must mark data in its response to be implicitly or explicitly cacheable or non-cacheable.
- Benefits
  - Performance improvement for client-side (latency, bandwidth)
  - Less load on the server
  - Hide network failures

Uniform Interface

- Uniform Interface constraints must satisfy these guidelines
  - Identification of resources (what is the **URI**? And consistency)
  - Manipulation of resources through **representations** (e.g. XML, JSON)
  - Self-descriptive messages
    - Request type and protocol: GET / HTTP/1.1
    - Protocol and response status: HTTP/1.1 200 OK
    - Media Type: Content-Type: application/json
  - Hypermedia as the engine of application state

Layered System

- Allows for a layer system architecture
  - Can break a system functionalities or services into several layers
  - For example, deploy APIs on server A, data storage on server B, identity management on server C.
  - Client cannot tell which system is directly connected

Code on demand (optional)

- Most of the time, the application sends static representations of resources like XML or JSON. However, when you need to, you are free to return executable code to support a part of your application
  - e.g., clients may call your API to get a UI widget rendering code
  - Or Java applets download

REST API Design concerns

- When should URI path segments be named with plural nouns?
- Which request method should be used to update resource state?
- How do I map non-CRUD operations to my URIs?
- What is the appropriate HTTP response status code for a given scenario?
- How can I manage the versions of a resource's state representations?
- How should I structure a hyperlink in JSON?

What is a Resource?

- The key abstraction of information in REST is a **resource**
- Any information that we can name can be a resource
- Examples?

Books

Customer

weather

question

User history

Music file

Football leagues

Users

location

answer

tweet

country

Resource Archetypes

A REST API is composed of 4 distinct resource archetypes

- Document
- Collection
- Store
- Controller

Resource Archetypes

- A **document** resource is a singular concept that is related to an object instance or database record. It usually includes both fields with values and links to other related resources.

  http://api.soccer.restapi.org/leagues/champions-league
  http://api.soccer.restapi.org/leagues/champions-league/teams/arsenal
  http://api.soccer.restapi.org/leagues/champions-league/teams/arsenal/players/ødegaard

- A **collection** resource is a server-managed directory of resources.

  http://api.soccer.restapi.org/leagues
  http://api.soccer.restapi.org/leagues/champions-league/teams
  http://api.soccer.restapi.org/leagues/champions-league/teams/arsenal/players

Resource Archetypes

- A **Store** is a client-managed resource repository. A store resource lets an API client put resources in, get them back out, and decide when to delete them.

```
PUT /users/1234/favorites/alonso
```

- A **Controller** resource models a procedural concept. Controller resources are like executable functions, with parameters and return values; inputs and outputs.

```
POST /alerts/245743/resend
```

Resource Representations

- A resource can be represented in different formats.
- It is the job of the server to support all the required representations
- Media Types
  - application/json, application/xml, text/html
  - image/gif, video/mpeg
- Client can specify which Media Type for the resource representation using the Accept header
  - Accept: application/json

Resource Representations

- Clients that consume REST API needs to know how the payload is encoded
  - i.e. Content Type
- 3 popular patterns
  - Using HTTP headers (e.g. *Content-Type:* and *Accept:*)
  - Using GET parameters (e.g. *&format=json*)
  - Using resource label (e.g. */foo.json*)

URI Path Design

Rule: A singular noun should be used for **document** names

http://api.soccer.restapi.org/leagues/champions-league

Rule: A plural noun should be used for **collection** names

http://api.soccer.restapi.org/leagues

Rule: A plural noun should be used for **store** names

PUT /users/1234/favorites

Rule: A verb or verb phrase should be used for **controller** names

POST /alerts/245743/resend

Rule: CRUD function names should not be used in URIs

Anti-pattern: POST /users/1234/delete

URI Format

URI = scheme "://" authority "/" path ["?" query] ["#" fragment]

Example - http://api.spotify.com/v1/artists/{id}/related-artists

scheme = HTTP
authority = api.spotify.com
path = v1/artists/{id}/related-artists

Rule: The query component of a URI may be used to filter collections or stores

GET /users?role=admin

Rule: The query component of a URI should be used to paginate collection or store results

GET /users?pageSize=25&pageStartIndex=50

HTTP Methods (Verbs)

- HTTP offers a set of methods for performing different operations

  - GET
  - PUT
  - POST
  - DELETE
  - HEAD
  - OPTIONS
  - TRACE
  - PATCH

GET

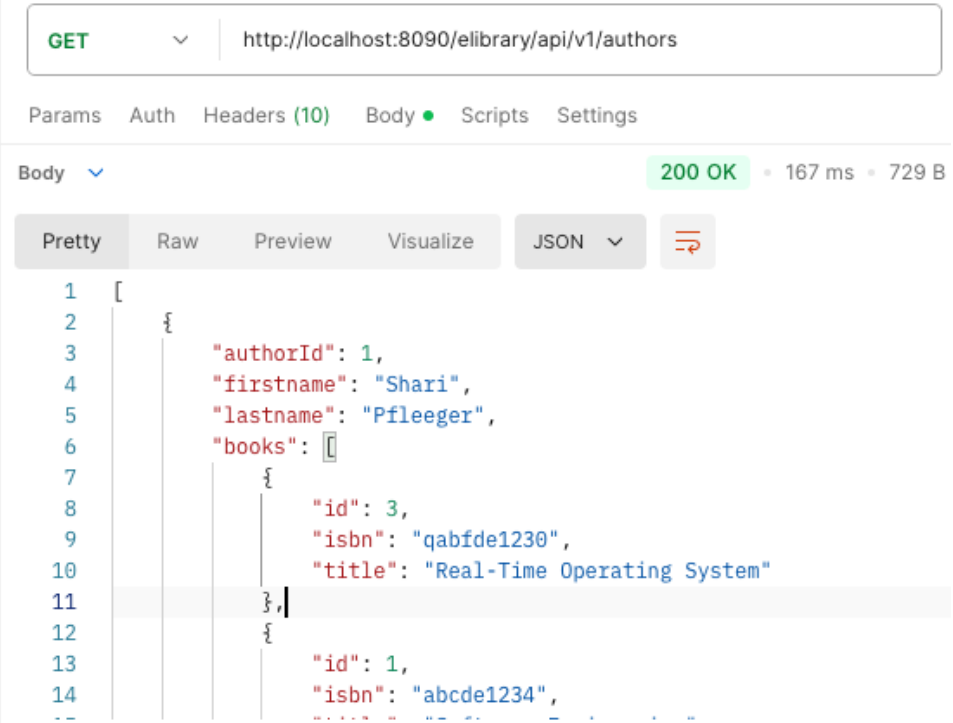
Retrieve a representation of a resource

Request

```
curl http://localhost:8090/elibrary/api/v1/authors
```

Response


```
[{"authorId": 4,"firstname": "Keith","lastname": "Ross"},
{"authorId": 1,"firstname": "Shari","lastname": "Pfleeger"},
{"authorId": 2,"firstname": "Perry","lastname": "Lea"},
{"authorId": 3,"firstname": "James","lastname": "Kurose"},
{"authorId": 5,"firstname": "Martin","lastname": "Kleppmann"}]
```

POST

Use POST to create a new resource in a **collection**

```
curl -X POST http:/localhost:8090/elibrary/api/v1/authors -d
'{"firstname":"Adam","lastname":"Shostack"}'
```

Use POST to execute **controllers**

```
POST /alerts/245743/resend
```

```
POST /convert?from=EUR&to=CNY&amount=100
```

- Use **PUT** to both insert and update a resource in a **store/collection**
  - Add a new resource to a store/collection
  - Update or replace an already stored resource

- use **DELETE** to request that a resource be completely removed from its parent, which is often a collection or store

```
DELETE /accounts/4ef2d5d0-cb7e-11e0-9572-0800200c9a66/buckets/objects/4321
```

- **HEAD**: Retrieve metadata-only representation
  - Retrieve response headers

```
curl --head http://localhost:8090/elibrary/api/v1/authors
```

```
HTTP/1.1 200
Content-Type: application/json
Content-Length: 277
Date: Tue, 19 Sep 2023 15:21:18 GMT
```

OPTIONS

Check which HTTP methods a particular resource supports

```
curl -X OPTIONS http://localhost:9091/library/api/v1/authors
```


```
<?xml version="1.0" encoding="UTF-8"?>
<ns0:application xmlns:ns0="http://wadl.dev.java.net/2009/02">
   <ns0:doc xmlns:ns1="http://jersey.java.net/" ns1:generatedBy="Jersey: 3.1.3 2023-07-21 16:14:39"/>
   <ns0:grammars>
      <ns0:include href="http://localhost:9091/library/api/v1/application.wadl/xsd0.xsd">
         <ns0:doc title="Generated" xml:lang="en"/>
      </ns0:include>
   </ns0:grammars>
   <ns0:resources base="http://localhost:9091/library/api/v1/">
      <ns0:resource path="authors">
         <ns0:method id="getAuthors" name="GET">
            <ns0:response>
               <ns0:representation mediaType="application/json"/>
            </ns0:response>
         </ns0:method>
         <ns0:method id="addAuthor" name="POST">
            <ns0:request>
               <ns0:representation element="author" mediaType="application/json"/>
            </ns0:request>
            <ns0:response>
               <ns0:representation mediaType="*/*"/>
            </ns0:response>
         </ns0:method>
      </ns0:resource>
   </ns0:resources>
</ns0:application>
```

Safety and Idempotence

- The GET, HEAD, OPTIONS, and TRACE methods are considered **safe** methods.
  - Request methods are considered **safe** if their defined semantics are essentially **read-only**.
  - The client does not request, and does not expect, any state change on the origin server as a result of applying a safe method to a target resource.
- GET, HEAD, **PUT, DELETE**, OPTIONS and TRACE are all intended to be **idempotent** operations
  - i.e., the side-effects of N > 0 identical requests is the same as for a single request
  - However, PUT and DELETE are not safe (changes the state)
- POST is both **non-idempotent** and **not safe**

https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

Pagination, sorting and filtering

- Pagination
  - Using limit and offset
    - e.g. /dogs?limit=25&offset=50 (records 50 through 75)
- Filtering
  - Add optional fields in a comma-delimited list
    - e.g. /dogs?fields=name,color,location
- Sorting
  - Can introduce order and sort
    - e.g. /2.3/articles/{ids}/comments?**order**=desc&**sort**=creation&site=stackoverflow

HTTP Status code

Category	Description
1xx	Informational
2xx	Success
3xx	Redirection
4xx	Client Error
5xx	Server Error

Examples

- 100 (“Continue”) initial part of request received and client should continue
- 201 (“Created”) must be used to indicate successful resource creation
- 202 (“Accepted”) must be used to indicate successful start of an asynchronous action
- 307 (“Temporary Redirect”) should be used to tell clients to resubmit the request to another URI
- 401 (“Unauthorized”) must be used when there is a problem with the client’s credentials
- 405 (“Method Not Allowed”) must be used when the HTTP method is not supported
- 500 (“Internal Server Error”) should be used to indicate API malfunction

Versioning

Let's look at some practices

- URI Versioning
  - http://api.example.com/v1
  - http://apiv1.example.com
- Versioning using custom request header
  - Accept: application/vnd.example.v1+json
  - Accept: application/vnd.example+json;version=1.0
- Versioning using "Accept-version" header
  - Accept-version: v1
  - Accept-version: v2

Summary: HTTP Method – Collection

HTTP Method	URL Design	Description
GET	api.hvl.no/library/v1/books	Get list of collection/store
POST	api.hvl.no/library/v1/books	Create a document
PUT	api.hvl.no/library/v1/books	Update/replace entire collection (Not often desirable)
DELETE	api.hvl.no/library/v1/books	Delete the entire collection (Not often desirable)

Summary: HTTP Method – Document

HTTP Method	URL Design	Description
GET	api.hvl.no/library/v1/books/{isbn}	Get one resource document
~~POST~~	~~api.hvl.no/library/v1/books/{isbn}~~	N/A
PUT	api.hvl.no/library/v1/books/{isbn}	Update a single resource
DELETE	api.hvl.no/library/v1/books/{isbn}	Delete a single resource

HTTP Method	/books	/books/{isbn}
GET	200 (OK). List of books. Can use pagination, sorting, and/or filtering.	200 (OK). Single book. 404 (Not Found), if isbn not found or invalid.
POST	201 (created) 'Location' header with link to /books/{isbn} containing new isbn.	404 (Not Found).
PUT	405 (Method Not Allowed), unless you want to update every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if isbn not found or invalid.
DELETE	405 (Method Not Allowed), unless you want to update every resource in the entire collection.	200 (OK). 404 (Not Found), if isbn not found or invalid.

Lab – REST Services

- Build on the previous library service
  - Create a RESTful web service for the library model
  - Spring Framework + REST
- You will get a startcode
- And we will build the resource model together in the next lecture

More resources

- Masse, Mark. *REST API design rulebook: designing consistent RESTful web service interfaces*. " O'Reilly Media, Inc.", 2011.
- https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- https://restfulapi.net/
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
- https://pages.apigee.com/rs/apigee/images/api-design-ebook-2012-03.pdf