COMP519 Web Programming
Lecture 27: REST
Handouts

Ullrich Hustadt

Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool

Contents

Web Services: Motivation

- There is a lot of data that is of interest to a multitude of parties

  Examples:

  - Addresses for a given postcode
  - Pictures associated with certain criteria such as location, year, or name

- It is in the interest of the owner/collator of that data to make access easy and automatable, possibly in exchange for payment
- Often it is also useful to be able to exchange data in both directions

  Examples:

  - A credit score company that provides credit scores for individuals to banks and receives information on loans back
  - A picture database that provides access to pictures but also allows photographers to upload pictures

- It is in the interest of the owner/collator of that data to make the submission of additional data easy and automatable

Web Services and Representational State Transfer

Web Service

A software system designed to support interoperable machine-to-machine interaction over the world wide web

Representational State Transfer (REST)

A software architectural style used in the creation of web services expressed as six constraints on the elements of a software architecture

RESTful Web Service

A web service with a REST architecture

Representational State Transfer

- Client-Server:
  - A server providing a set of services listens to requests for these services
  - A client that wants to use a particular service sends a request to the server
  - The server can then either reject or execute the requested service and return a response to the client
- Stateless:
  Communication between client and server must be done without storing any type of state on the server
  $\rightsquigarrow$ every client request must hold all information necessary for it to be processed
- Cacheability:
  - Responses must implicitly or explicitly indicate whether they are cacheable or non-cachable
  - Clients and intermediaries can cache cachable responses and reuse cached responses for later equivalent requests

Representational State Transfer

- Layered system
  - A client is ignorant whether it is connected directly to a server or to an intermediary
  - Intermediaries might add a security layer on top of a web service
  - A server may itself act as a client to multiple other servers to generate a response to a client
- Code on demand:
  Servers can temporarily extend or customize the functionality of a client by transferring executable code, for example, JavaScript, to it

Representational State Transfer

- Uniform interface:
  - Resource identification in requests
    - REST treats data / content as resources and collections of resources
    - Resources are identified using Unique Resource Identifiers (URIs) in a request
    - The resources themselves are conceptually separate from the representations that are returned to the client
  - Resource manipulation through representations
    When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource
  - Self-descriptive requests / responses
    Each request and each response includes enough information to describe how to process it
  - Hypermedia as the engine of application state (HATEOAS)
    Having accessed an initial URI for a RESTful web service, a client should be able to use server-provided links to dynamically discover all the available services

Example

- We want to provide a web service that maintains information on Computer Science students at the University of Liverpool
- The web service should allow us to
  - Retrieve information on all students
  - Retrieve information on a specific student (via student id)
  - Add information on a new student
  - Modify information on an existing student
  - Delete (information on) a student

RESTful Web Services: Resources and URIs

- A resource can be a singleton or a collection
- A resource may contain sub-collections of resources
  $\rightsquigarrow$ resources form a hierarchical structure

– `students`
  is a collection of all students
– `students/201912345`
  is a singleton student in that collection
– `students/201912345/addresses`
  is a sub-collection of addresses for a student
– `students/201912345/addresses/tAddr`
  is a singleton address in that sub-collection

RESTful Web Services: Resources and URIs

- A resource can be a singleton or a collection
- A resource may contain sub-collections of resources
  $\rightsquigarrow$ resources form a hierarchical structure
  - `students/201912345`
    is a singleton student in the `students` collection
  - `students/201912345/addresses/tAddr`
    is a singleton address among all addresses of a student
- A REST API uses Uniform Resource Identifiers (URIs) to address resources
  - https://api.liv.ac.uk/students/201912345
    is the URI for the student with id 201912345
  - https://api.liv.ac.uk/students/201912345/addresses/tAddr
    is the URI for that student's term time address
    provided https://api.liv.ac.uk is the base URL for our web service

Naming Conventions for Resources

- Include a version number either into the base URL or make it the top-level of the resource hierarchy

```
https://v1.api.liv.ac.uk/students/
https://api.liv.ac.uk/v1/students/
```

- Use nouns, not verbs, to represent resources
  $\rightsquigarrow$ use of identification numbers is permissible

```
students not getStudents
```

- Use singular nouns for singleton resources

```
tAddr not tAddrs
```

- Use plural nouns for collections of resources

```
students not student
```

- Use camelCasing for compound nouns

```
tAddr not term-time nor t_addr
```

- Use forward slash (/) to indicate a hierarchical relationship

```
students/201912345 not students-201912345
```

RESTful Web Services: Operations and HTTP Methods

- A REST API associates a specific HTTP method with each specific operation that can be performed on a resource
- The association should reflect the generic meaning of a HTTP method

  'Retrieve information' is associated with `GET`
  'Delete information' is associated with `DELETE`

- A REST API uses HTTP response codes to indicate each specific outcome of a request

  `200 (OK)` is associated with a successful operation
  `404 (NOT FOUND)` is associated with a failure to find a resource

RESTful Web Services: Operations and HTTP Methods

- The generic meaning of a HTTP method depends on whether the URI involved is for
  - a collection of resources
  - a singleton / individual resource

  As part of a collection, an individual resource is also called member resource

- Properties of requests and responses that are of interest:
  - Cachable response:
    Clients and intermediaries can cache cachable responses and reuse cached responses for later equivalent requests
  - Safe request:
    A safe request does not change the state of the resource
  - Idempotent request:
    Repeating an idempotent request must produce the same effect on the server (but not necessarily the same response)

RESTful Web Services and HTTP Methods

GET on a member resource (cacheable, idempotent, safe)
Retrieve a representation of the resource Response codes: 200 (OK), 404 (NOT FOUND)

Request:

```
GET /students/2019123456/addresses/tAddr HTTP/1.1
Host: v1.api.liv.ac.uk
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{"streetHN":"1 Abby Road", "city":"Liverpool",
 "postCode":"L69 9AA", "country":"UK"}
```

RESTful Web Services and HTTP Methods

GET on a member resource (cacheable, idempotent, safe)
Retrieve a representation of the resource Response codes: 200 (OK), 404 (NOT FOUND)

Request:
`GET /students/2050123456 HTTP/1.1`
`Host: v1.api.liv.ac.uk`
 	
Response:
`HTTP/1.1 404 NOT FOUND`

The HTTP response code 404 in the example above indicates that a student with that student id does not exist (assuming we know that the collection `students` exists)

RESTful Web Services and HTTP Methods

GET on a member resource (cacheable, idempotent, safe)
Retrieve a representation of the resource Response codes: 200 (OK), 404 (NOT FOUND)

Request:
`GET /students/2019123000/addresses/tAddr HTTP/1.1`
`Host: v1.api.liv.ac.uk`
 	
Response:
`HTTP/1.1 404 NOT FOUND`

Same HTTP response code but fault is not immediately obvious (student with id 2019123000 may not exist or the student exists but has no term time address)

GET on Complex Member Resources

If a member resource M 'contains' a collection or other member resource, we can represent M as follows:

- via embedding: A full representation of the collections and member resources is embedded in the representation of M

```
{"id":"2019123456", "sname":"Ady", "fname":"Ada",
  "addresses":
  ["tAddr":{"streetHN":"1␣Abby␣Road", "city":"Liverpool",
            "postCode":"L69␣9AA", "country":"UK"},
   "pAddr":{"streetHN":"9␣Mort␣Street", "city":"Wigan",
            "postCode":"WN2␣4TU", "country":"UK"}]}
```

This becomes problematic if either the collections are very large or we are trying to model a circular relationship, for example, if we try to add an attribute "tutor" for a member resource that in turn contains a collection "tutees"

GET on Complex Member Resources

If a member resource M 'contains' a collection or other member resource, we can represent M as follows:

- via links: Links to other resources are embedded

```
{"id":"2019123456", "sname":"Ady", "fname":"Ada",
 "_links":[{"href":"students/2019123456/addresses",
            "method":"GET","rel":"addresses"},
           {"href":"students/2019123456",
            "method":"GET","rel":"self"}]}
```

  This provides the most 'compact' representation

- via relationships: Links to each member resource of each collection are embedded

```
{"id":"2019123456", "sname":"Ady", "fname":"Ada",
 "relationships": {
   "addresses":["students/2019123456/addresses/tAddr",
                "students/2019123456/addresses/pAddr"]}}
```

  This is most appropriate if those member resources exist independently, for example, modules, academic advisors (but not addresses)

RESTful Web Services and HTTP Methods

GET on a member resource (cacheable, idempotent, safe)
Retrieve a representation of the resource Response codes: 200 (OK), 404 (NOT FOUND)

Request:

```
GET /students/2019123456 HTTP/1.1
Host: v1.api.liv.ac.uk
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{"id":"2019123456", "sname":"Ady", "fname":"Ada",
 "_links":[{"href":"students/2019123456/addresses",
            "method":"GET","rel":"addresses"},
           {"href":"students/2019123456",
            "method":"GET","rel":"self"}]}
```

RESTful Web Services and HTTP Methods

DELETE on a member resource (*cacheable*, idempotent, not safe)

Delete that member resource

Response codes: 200 (OK), 204 (NO CONTENT), 404 (NOT FOUND)

Request:

```
DELETE /students/2019123456 HTTP/1.1
Host: v1.api.liv.ac.uk
```

Response:

```
HTTP/1.1 204 NO CONTENT
```

- HTTP response code 204 indicates that deletion was successful and that there is no response body
- HTTP response code 200 indicates that deletion was successful but we also send data back, e.g., a representation of what was deleted
- HTTP response code 404 can be used if the member resource did not exist

RESTful Web Services and HTTP Methods

POST on a member resource (not cacheable, not idempotent, not safe)

- Create a new specific member resource using the data in the request body
- The URI of the member resource is returned in the response Location header field

R'Cs: 201 (CREATED), 400 (BAD REQUEST), 409 (CONFLICT)

Request:

```
POST /students/2019123458/addresses/tAddr HTTP/1.1
Host: v1.api.liv.ac.uk
Content-Type: application/json; charset=utf-8

{"streetHN":"3 Carl's Court", "city":"Liverpool",
 "postCode":"L69 3CC","country":"UK"}}
```

Response:

```
HTTP/1.1 201 CREATED
Location: https://v1.api.liv.ac.uk/students/2019123458/
          ↪addresses/tAddr
```

RESTful Web Services and HTTP Methods

POST on a member resource (not cacheable, not idempotent, not safe)

- Create a new specific member resource using the data in the request body
- The URI of the member resource is returned in the response Location header field

R'Cs: 201 (CREATED), 400 (BAD REQUEST), 409 (CONFLICT)

Request:

```
POST /students/2019123458/addresses/tAddr HTTP/1.1
Host: v1.api.liv.ac.uk
Content-Type: application/json; charset=utf-8

{"streetHN":"1 Abby Road", "city":"Liverpool"}
```

Response:

```
HTTP/1.1 400 BAD REQUEST
```

HTTP response code 400 indicates that not enough or incorrect data was provided for the new member resource, e.g., no post code was provided

RESTful Web Services and HTTP Methods

POST on a member resource (not cacheable, not idempotent, not safe)

- Create a new specific member resource using the data in the request body
- The URI of the member resource is returned in the response Location header field

R'Cs: 201 (CREATED), 400 (BAD REQUEST), 409 (CONFLICT)

Request:

```
POST /students/2019123457/addresses/tAddr HTTP/1.1
Host: v1.api.liv.ac.uk
Content-Type: application/json; charset=utf-8

{"streetHN":"1 Abby Rd","city":"Lvpl","postCode":"L69 9AA"}
```

Response:

```
HTTP/1.1 409 CONFLICT
```

HTTP response code 409 indicates that the new member resource conflicts with an already existing one, e.g., the student already had a term time address

RESTful Web Services and HTTP Methods

PUT on a member resource (not cacheable, idempotent, not safe)

If the member resource exists,

– replace it with one based on the information in the request body

If the member resource does not exist,

– behave like POST on a member resource

R'Cs: 200 (OK) + response codes for POST

Request:

```
PUT /students/2019123456/addresses/tAddr HTTP/1.1
Host: v1.api.liv.ac.uk
Content-Type: application/json; charset=utf-8

{"streetHN":"2 Bank Lane", "city":"Liverpool",
 "postCode":"L69 2BB", "country":"UK"}
```

Response:

```
HTTP/1.1 200 OK
```

HTTP response code 200 indicates that the member resource was successfully replaced and we send the modified member resource back

RESTful Web Services and HTTP Methods

`PUT` on a member resource (not cacheable, idempotent, not safe)

If the member resource exists,

– replace it with one based on the information in the request body

If the member resource does not exist,

– behave like `POST` on a member resource

R'Cs: 200 (OK) + response codes for `POST`

Request:

```
PUT /students/2019123456/addresses/tAddr HTTP/1.1
Host: v1.api.liv.ac.uk
Content-Type: application/json; charset=utf-8

{"streetHN":"2 Bank Lane", "city":"Liverpool"}
```

Response:

```
HTTP/1.1 400 BAD REQUEST
```

HTTP response code 400 indicates that not enough or incorrect data was provided for the update, e.g., no post code was provided

RESTful Web Services and HTTP Methods

`PATCH` on a member resource (not cacheable, idempotent, not safe)
If the member resource exists, – update parts of the member resource using the data in the request body If the member resource does not exist, – report an error or create the member resource using the data in the request body Response codes: `200 (OK)`, `404 (NOT FOUND)`

RESTful Web Services and HTTP Methods

PATCH on a member resource (not cacheable, idempotent, not safe)

If the member resource exists,

– update parts of the member resource using the data in the request body

Response codes: 200 (OK), 404 (NOT FOUND)

Request:

```
PATCH /students/2019123456/addresses/tAddr HTTP/1.1
Host: v1.api.liv.ac.uk
Content-Type: application/json; charset=utf-8

{"streetHN":"2 Bank Lane", "postCode":"L69 2BB"}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{"streetHN":"2 Bank Lane", "city":"Liverpool",
 "postCode":"L69 2BB", "country":"UK"}
```

RESTful Web Services and HTTP Methods

PATCH on a member resource (not cacheable, idempotent, not safe)

If the member resource exists,

– update parts of the member resource using the data in the request body

Response codes: 200 (OK), 404 (NOT FOUND)

Request:

```
PATCH /students/2019123456/addresses/tAddr HTTP/1.1
Host: v1.api.liv.ac.uk
Content-Type: application/json; charset=utf-8

{"streetHN":"2 Bank Lane", "postCode":"L69 2BB"}
```

Response:

```
HTTP/1.1 200 OK
...
```

HTTP response code 200 indicates that the member resource was successfully updated with data in the request body

Note: Incomplete data is fine

RESTful Web Services and HTTP Methods

PATCH on a member resource (not cacheable, idempotent, not safe)

If the member resource does not exist,

- report an error or create the member resource using the data in the request body

Response codes: 200 (OK), 404 (NOT FOUND)

Request:

```
PATCH /students/2019123456/addresses/pAddr HTTP/1.1
Host: v1.api.liv.ac.uk
Content-Type: application/json; charset=utf-8

{"streetHN":"2 Bank Lane", "postCode":"L69 2BB"}
```

Response:

```
HTTP/1.1 404 NOT FOUND
```

HTTP response code 404 indicates that the member resource did not exist and therefore could not be updated

RESTful Web Services and HTTP Methods

`POST` on a collection (not cacheable, not idempotent, not safe)

- Create a new member resource in the collection using the information in the request body
- The URI of the created member resource is automatically assigned and returned in the response Location header field

Response codes: `201 (CREATED)`, `400 (BAD REQUEST)`

Request:

```
POST /students HTTP/1.1
Host: v1.api.liv.ac.uk
Content-Type: application/json; charset=utf-8

{"sname":"Clay","fname":"Cia",
 "tAddr":{"streetHN":"3 Carl's Court", "city":"Liverpool",
 "postCode":"L69 3CC","country":"UK"}}
```

Response:

```
HTTP/1.1 201 CREATED
Location: https://v1.api.liv.ac.uk/students/2019123458
```

RESTful Web Services and HTTP Methods

POST on a collection (not cacheable, not idempotent, not safe)

– Create a new member resource in the collection using the information

Response codes: 201 (CREATED), 400 (BAD REQUEST)

Request:

```
POST /students HTTP/1.1
Host: v1.api.liv.ac.uk
Content-Type: application/json; charset=utf-8

{"sname":"Deng",
 "tAddr":{"streetHN":"3 Carl's Court", "city":"Liverpool",
 "postCode":"L69 3CC","country":"UK"}}
```

Response:

```
HTTP/1.1 400 BAD REQUEST
```

HTTP response code 400 indicates that not enough or incorrect data was provided, e.g., no first name was provided

How much data is 'enough' will depend on the web service

RESTful Web Services and HTTP Methods

DELETE on a collection (not cacheable, idempotent, not safe)

Delete all member resources of the collection but not the collection

Response codes: 200 (OK), 204 (NO CONTENT), 404 (NOT FOUND)

Request:

```
DELETE /students HTTP/1.1
Host: v1.api.liv.ac.uk
```

Response:

```
HTTP/1.1 204 NO CONTENT
```

- HTTP response code 204 indicates that deletion was successful and that there is no response body
- HTTP response code 200 indicates that deletion was successful but we also send data back, e.g., a representation of what was deleted
- HTTP response code 404 can be used if the collection did not exist

RESTful Web Services and HTTP Methods

GET on a collection (cacheable, idempotent, safe)
Retrieve the URIs or representations of all the member resources of the collection Response codes: 200 (OK), 404 (NOT FOUND)

Request:

```
GET /students HTTP/1.1
Host: v1.api.liv.ac.uk
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[{"id":"2019123456", "sname":"Ady", "fname":"Ada",
  "tAddr":{"streetHN":"1 Abby Road", "city":"Liverpool",
  "postCode":"L69 9AA"}},
 {"id":"2019123457", "sname":"Bain", "fname":"Ben",
  "tAddr":{"streetHN":"2 Bank Lane", "city":"Liverpool",
  "postCode":"L69 2BB"}}]
```

RESTful Web Services and HTTP Methods

GET on a collection (cacheable, idempotent, safe)
Retrieve the URIs or representations of all the member resources of the collection Response codes: 200 (OK), 404 (NOT FOUND)

Request:

```
GET /students HTTP/1.1
Host: v1.api.liv.ac.uk
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[{"id":"2019123456","sname":"Ady","fname":"Ada","_links":
  [{"href":"students/2019123456/addresses","method":"GET",
    "rel":"addresses"},{"href":"students/2019123456",
    "method":"GET","rel":"self"}]},
 {"id":"2019123457","sname":"Bain","fname":"Ben","_links":
   [{"href":"students/2019123457/addresses","method":"GET",
     "rel":"addresses"}, ...]}]
```

RESTful Web Services and HTTP Methods

GET on a collection (cacheable, idempotent, safe)
Retrieve a representation of the resource Response codes: 200 (OK), 404 (NOT FOUND)

Request:
`GET /academics HTTP/1.1`
`Host: v1.api.liv.ac.uk`
 	
Response:
`HTTP/1.1 404 NOT FOUND`

The HTTP response code 404 in the example above indicates that the collection `academics` does not exist

Query Parameters: Filtering / Sorting

RESTful Web Services often allow the response to a GET request on a collection to be modified by query parameters

`GET /students?sort=sname`
returns the list of all students ordered by their surname
`GET /students?sname_contains=Smith`
returns the list of all students whose surname contains the substring 'Smith'
`GET /students?sname_contains=Smith&sort=id`
returns the list of all students whose surname contains the substring 'Smith' ordered by student ID

Query Parameters: Pagination

Pagination can be used to return a representation of only a part of a collection in response to a GET request

```
Request:
GET /students?page=2 HTTP/1.1
Host: v1.api.liv.ac.uk

Response:
HTTP/1.1 200 OK
Content-Type: application/json

{ "page_number": 2,
  "total_pages": 154,
  "total_count": 15342,
  "previous_page": "/students?page=1",
  "next_page": "/students?page=3",
  "last_page": "/students?page=154",
  "results": [ ... ] }
```

HATEOAS Revisited

HATEOS

Having accessed an initial URI for a RESTful web service, a client should be able to use server-provided links to dynamically discover all the available services

Following this principle, data returned in response to requests should also include information about the actions that can be performed

HATEOAS Revisited

Accessing the 'root' of a web service should give us a list of links and HTTP request methods that we can use with this service

```
Request:
GET / HTTP/1.1
Host: v1.api.liv.ac.uk

Response:
HTTP/1.1 200 OK
Content-Type: application/json

{ "_links": [ { "href": "/students", "method": "GET",
                "rel":  "collection" },
              { "href": "/students", "method": "POST",
                "rel":  "edit" },
              { "href": "/students", "method": "DELETE",
                "rel":  "delete" }
            ] }
```

HATEOAS Revisited

Accessing a collection (member resource) should give us a list of links and HTTP request methods that we can use on that collection (member resource)

```
Request:
GET /students/2019123456/addresses/tAddr HTTP/1.1
Host: v1.api.liv.ac.uk

Response:
HTTP/1.1 200 OK
Content-Type: application/json

{"streetHN":"1 Abby Road", "city":"Liverpool",
 "postCode":"L69 9AA", "country":"UK",
 "_links": [{"href":"/students/2019123456/addresses/tAddr",
             "method": "GET",    "rel":  "self" },
            {"href":"/students/2019123456/addresses/tAddr",
             "method": "PATCH",  "rel":  "edit" },
            {"href":"/students/2019123456/addresses/tAddr",
             "method": "DELETE", "rel":  "delete" } ] }
```

Revision and Further Reading

Read

- Chapter 1: Introduction to REST
- Chapter 4: Resource-Oriented Services: Designing Services

of S. Abeysinghe: RESTful PHP Web Services.
Packt Publishing, 2008.