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

- The web service we want to provide 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 singleton student in that collection
- `students/201912345/addresses`
  is a sub-collection of addresses for a student
- `students/201912345/addresses/termTime`
  is a singleton address in that sub-collection

- 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 provided that
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

```
termTime not termTimes
```

- Use plural nouns for collections of resources

```
students not student
```

- Use camelCasing for compound nouns

```
termTime not term-time nor term_time
```

- 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 a HTTP response code 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:
  - Idempotent request:
    Repeating an idempotent request must produce the same response every time until another request changes the state of the resource
  - Safe request:
    A safe request does not change the state of the resource
  - Cachable response:
    Clients and intermediaries can cache cachable responses and reuse cached responses for later equivalent requests

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), 408 (BAD REQUEST)

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",
  "termTime":{"line1":"1 Abby Road", "city":"Liverpool",
  "postCode":"L69 9AA"}},
 {"id":"2019123457", "sname":"Bain", "fname":"Ben",
  "termTime":{"line1":"2 Bank Lane", "city":"Liverpool",
  "postCode":"L69 2BB"}}]
```

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)

Request:

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

{"sname":"Clay","fname":"Cia",
 "termTime":{"line1":"3 Carl's Court", "city":"Liverpool",
 "postCode":"L69 3CC"}}
```

Response:

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

RESTful Web Services and HTTP Methods

`DELETE` on a collection (not cacheable, idempotent, not safe)
Delete all member resources of the collection
Response codes: `200 (OK)`, `404 (NOT FOUND)`

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)`, `408 (BAD REQUEST)`
`DELETE` on a member resource (*cacheable*, idempotent, not safe)
Delete that member resource Response codes: `200 (OK)`, `404 (NOT FOUND)`

RESTful Web Services and HTTP Methods

`POST` on a member resource (not cacheable, not idempotent, not safe)
– Create a new member resource in the member resource using the instructions in the request body – The URI of the created member resource is automatically assigned and returned in the response Location header field
`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, – create a new one based on 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)`

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 instructions in the request body If the member resource does not exist, – report an error or create the member resource using the instructions in the request body Response codes: `200 (OK)`, `204 (NOT FOUND)`, `404 (NOT FOUND)`

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.