REST in Operation

CMPU4023 - Enterprise Applicaton Development

REST and HTTP

- Recall that, in REST, resources are the primary abstraction of service state and behaviour
- A RESTful API is implemented as a message passing interface which carries out some specific operation on some specific service-exposed resource
- Resources are exposed as RESTful endpoints (i.e. URIs)
- That implementation is closely tied to the features of HTTP
- Unlike an RPC-style API where many different kinds of parameterised commands can be defined, REST APIs implement relatively few commands, focusing on the so-called ***CRUD*** operations on resources

Items and Collections

- Resources fall broadly into two basic types

1. An **item** resource is a single resource used to represent some concrete or synthetic group of attributes, usually represented as key/value pairs (e.g. an invoice record in a database or monitored performance attributes of a server)

2. A **collection** resource is a group of item resources of the same item resource type

- Items and collections has slightly different semantics in a RESTful API so they support some but not all of the same messages as we will see
- Both types are uniquely identified by a URI

Idempotence

- Idempotence is the property of an operation such that the operation can be applied multiple times to some value without changing the outcome beyond its first application
- In the context of REST, we can say that some operations are idempotent if, after the first application of an operation on a resource (which may alter service state), subsequent applications of that operation don’t alter the service state, no matter how many times that operation is later applied
- Consideration of idempotence is important in the context of a message passing API over a latent, unreliable network as clients need guaranteed API semantics in the face of potential network partitions

Create Resource

- A new item resource is created within a collection resource
- This assumes the *a priori* existence of the collection itself giving rise to the following properties of collections

  - There is no create collection resource operation or requirement
  - Collections are themselves immutable though not their contained item resources

- The HTTP `POST` command is used to create item resources having a message body containing the attribute key/values to populate the newly created resource
- POST is not idempotent as each successful call creates a new resource

HTTP POST Example

- In this example, the customers collection, the customer item and the notices assocation collection already exist
- A new notice item resource is being created within the notices collection having the specified attributes

```
curl -X POST https://api.example.com/customers/237324632/notices -d
{
  "subject":  "Account status",
  "body":     "Dear john, please review your outstanding balance ...",
  "delivery": "urgent"
}
```

- By convention the return code is 201 (created) and a newly-created resource will be assigned a unique identifier which can be used in later

Read Resource

- An item resource or a collection resource can be read using their resource identifiers (URIs)
- Reading a resource should not, by side-effect, update the state of the resource
- In practice, many implementations maintain metadata such as last-read timestamp attributes associated with resources which would be updated
- The HTTP GET command is used to read resources and this is considered to be idempotent provided it is strictly implemented which respect to the resource state proper

HTTP GET Example

- The following example, the resource is addressed by its unique identifier and its contents are returned

```
curl -X GET https://api.example.com/customers/237324632/notices/213

HTTP/1.1 200 OK
{
  "subject":  "Account status",
  "body":     "Dear john, please review your outstanding balance ...",
  "delivery": "urgent"
}
```

- The return code is 200 (OK) if found or 404 (Not found) if unknown

Update Resource

- An already-existing resource can be updated in one of two ways, either fully or partially
- The HTTP PUT command is used to fully update a resource is considered to be an idempotent operation as multiple PUTs with the same values don't alter the state following the first one
- The HTTP PATCH command is used to partially update a resource but this is not considered to be automatically idempotent because of the way an API might do the updates
- An update which replaces an attribute in place would be idempotent but an operation to add to a list, for example, would not

HTTP PUT Example

- The following example, the resource is addressed by its unique identifier and its contents are fully replaced by the newly specified attributes

```
curl -X PUT https://api.example.com/customers/237324632/notices/213 -d
{
  "subject":  "Account status",
  "body":     "Dear john, please review your outstanding balance ...",
  "delivery": "normal"
}
```

- The updated resource retains its previously-assigned identifier following the full update
- The return code is 200 (OK) or one of the 4xx codes if an error occurs in the update

HTTP PATCH Example

- Once again, the resource is addressed by its unique identifier but its contents are only partially replaced by the newly specified attributes

```
curl -X PATCH https://api.example.com/customers/237324632/notices/213 -d
{
  "delivery": "normal"
}
```

- As before, updated resource retains its previously-assigned identifier following the full update
- This PATCH example is idempotent
- The return code is 200 (OK) or one of the 4xx codes if an error occurs in the update

Delete Resource

- An already-existing item resource can be removed from a collection resource
- The resource to be removed is identified by its URI as before
- The HTTP DELETE command is used to effect a resource removal and this is considered to be idempotent because once the item resource is removed it doesn’t matter how many subsequent requests to remove it are made
- Services may return a 404 (not found) error, a 410 (gone) error or 200 (OK) status depending on what what

HTTP DELETE Example

- The following example, the resource is addressed by its unique identifier and its contents are fully replaced by the newly specified attributes

```
curl -X DELETE https://api.example.com/customers/237324632/notices/213
```

- The updated resource retains its previously-assigned identifier following the full update
- The return code is 200 (OK) or one of the 4xx codes if an error occurs in the update

HTTP OPTIONS Command

- HTTP also supports an `OPTIONS` command which can, theoretically, be sent to an REST API endpoint to query characteristic of the associated resource
- For example, in self-documenting API, an `OPTIONS` command response could be used to explain what HTTP commands the endpoint supports, the request and response messages body structures, security requirements and so on
- However, the `OPTIONS` command is optional and is very often not implemented in practice
- We'll return to this theme later in the module

Summary

- A RESTful API is implemented as a message passing interface which carries CRUD operations on some specific service-exposed item or collection of items resource types identified by unique URIs
- REST operations are mapped on to the HTTP commands POST, GET, PUT, PATCH, DELETE and OPTIONS
- Operations are said to be idempotent if they can be carried out multiple times without changing the service state after the first one