RESTful APIs

Spring 2024

Web application servers as service providers

We've seen web application servers as backends to browser-based applications

- Browser is the client, making calls to the web application server for data

We can also use web application servers to provide a **service** to other programs

- Picture manipulation / storage service
- Book catalog service
- Git repository manipulation service

The web application server is just a vehicle to expose the service

RESTful service

REST (**REpresentational State Transfer**) is a methodology / architecture / philosophy for structuring distributed services

Based on Roy Fielding's 2000 dissertation:

Architectural Styles and the Design of Network-based Software Architectures

Web/HTTP 1.0 (1990) → Fielding (2000) → HTTP 1.1 (1999)

Ideas from REST have made it into HTTP, but HTTP is not REST

Resources

A RESTful service provide access to **resources** and **collections of resources**

Each resource has a unique "identifier" you use to access that resource

- **URI : Uniform Resource Identifier**

A collection of resources also has a unique "identifier" to access the collection

Hierarchical resources:

- resources can have sub-resources, themselves collected into collections

Example: kanban board

A kanban board has a collection of lists, with each list a collection of cards, each card containing something useful.

Resources: **list**, **card**

URI for collection of all lists:	`/lists`
URI for list 234:	`/lists/234`
URI for collection of cards in list 234:	`/lists/234/cards`
URI for card 56 in list 234:	`/lists/234/cards/56`

CRUD operations

There are four basic operations you can perform on resources in general

- **create** → add a resource to a collection
- **read** → return information on the resource
- **update** → change information in the resource
- **delete** → delete resource from a collection

Relational database tables, say, provide CRUD operations on the table via SQL

A RESTful service should allow you to perform those operations on its resources

RESTful services over HTTP

Web application server manages the resources and provides access/operations

A URI is a **URL** (hostname + local name of resource)

CRUD operations are implemented using different HTTP methods on the URL of the resource being operated on:

- create → **POST**
- read → **GET**
- update → **PUT** / **PATCH**
- delete → **DELETE**

Example: RESTful API for kanban board

```
   GET    /lists
  POST    /lists

   GET    /lists/{:listId}
   PUT    /lists/{:listId}
DELETE   /lists/{:listId}

   GET    /lists/{:listId}/cards
  POST    /lists/{:listId}/cards

   GET    /lists/{:listId}/cards/{:cardId}
   PUT    /lists/{:listId}/cards/{:cardId}
DELETE   /lists/{:listId}/cards/{:cardId}
```

Example: RESTful API for kanban board

```
   GET    /lists
  POST    /lists

   GET    /lists/{:listId}
   PUT    /lists/{:listId}
DELETE    /lists/{:listId}

   GET    /lists/{:listId}/cards
  POST    /lists/{:listId}/cards

   GET    /lists/{:listId}/cards/{:car
   PUT    /lists/{:listId}/cards/{:car
DELETE    /lists/{:listId}/cards/{:car
```

POST / PUT: Body contains the resource to create or update

Responses for POST should contain URI of created resource

Response for collection GET should contain the list of URIs of resources in the collection

(Or enough info to construct URI)

Implementation

Assign a route to each URI

The same route can be accessed by different HTTP methods

- can have have a function associated to the route for each method

The code for each route should:

- update data structures for the resources denoted by the route