Full Stack Application Development

REST Design

Akshaya Ganesan

Agenda

- Interaction Design with HTTP(response codes, request methods)
- Identifier Design with URIs
- Metadata Design(Media Types, content negotiation)
- Representation Design(Message body format, Hypermedia Representation)

Source: Font: Arial, Font Size: 8 pts

REST API Design

- API design is more of an art.
- Some best practices for REST API design are implicit in the HTTP standard.
  - When should URI path segments be named with plural nouns?
  - Which request method should be used to update the 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?

API Design

- REST APIs are designed around *resources* , which is any kind of object, data, or service that is accessible to the client.
- A resource has an *identifier* , a URI that uniquely identifies that resource.
- The resource URIs should be based on nouns (the resource) and not verbs (the operations on the resource).

API Design Elements

- The following aspects of API design are all important, and together they define your API:
  - The representations of your resources and the links to related resources.
  - The use of standard (and occasionally custom) HTTP headers.
  - The URLs and URI templates define your API's query interface for locating resources based on their data.
  - Required behaviors by clients—for example, DNS caching behaviors, retry behaviors

- Interaction Design with HTTP(response codes, request methods)
- Identifier Design with URIs
- Metadata Design(Media Types, content negotiation)
- Representation Design(Message body format, Hypermedia Representation)

Interaction Design

- Interaction Design with HTTP
- REST APIs embrace all aspects of the HyperText Transfer Protocol including its request methods, response codes, and message headers
- Each HTTP method has specific, well-defined semantics within the context of a REST API’s resource model.
- The purpose of GET is to retrieve a representation of a resource’s state.
- HEAD is used to retrieve the metadata associated with the resource’s state.
- PUT should be used to add a new resource to a store or update a resource.
- DELETE removes a resource from its parent.
- POST should be used to create a new resource within a collection and execute controllers.

Interaction Design

- Interaction Design with HTTP

- Rule: GET and POST must not be used to tunnel other request methods
- Rule: GET must be used to retrieve a representation of a resource
- Rule: HEAD should be used to retrieve response headers
- Rule: PUT must be used to both insert a stored resource
- Rule: POST must be used to create a new resource in a collection
- Rule: POST must be used to execute controllers
- Rule: DELETE must be used to remove a resource from its parent
- Rule: OPTIONS should be used to retrieve metadata that describes a resource's available interactions

Interaction Design

HTTP response success code summary

- 200 OK Indicates a nonspecific success
- 201 Created Sent primarily by collections and stores but sometimes also by controllers to indicate that a new resource has been created
- 204 No Content Indicates that the body has been intentionally left blank
- 301 Moved Permanently Indicates that a new *permanent* URI has been assigned to the client's requested resource
- 303 See Other Sent by controllers to return results that it considers optional
- 401 Unauthorized Sent when the client either provided invalid credentials or forgot to send them
- 402 Forbidden Sent to deny access to a protected resource
- 404 Not Found Sent when the client tried to interact with a URI that the REST API could not map to a resource
- 405 Method Not Allowed Sent when the client tried to interact using an unsupported HTTP method
- 406 Not Acceptable Sent when the client tried to request data in an unsupported media type format
- 500 Internal Server Error Tells the client that the API is having problems of its own

Identifier Design

- REST APIs use Uniform Resource Identifiers (URIs) to address resources
- URI Format
- URI = scheme "://" authority "/" path ["?" query] ["#" fragment]

- Forward slash separator (/) must be used to indicate a hierarchical relationship.
- Consistent subdomain names should be used for your APIs

Identifier Design

Resource Modelling

The URI path conveys a REST API's resource model,

Each forward slash-separated path segment corresponds to a unique resource within the model's hierarchy.

http://api.library.restapi.org/books/harry-potter

http://api.library.restapi.org/books

http://api.library.restapi.org/sections/fiction

http://api.library.restapi.org/sections

Identifier Design

document,

http://api.library.restapi.org/books/Pride and Prejudice

http://api.library.restapi.org/books/harry-potter/harry potter and the philosopher's stone

collection,

Each URI below identifies a collection resource:

http://api.library.restapi.org/books

http://api.library.restapi.org/sections

controller.

POST /alerts/245743/resend

Few antipatterns

GET /deleteUser?id=1234

GET /deleteUser/1234

DELETE /deleteUser/1234

POST /users/1234/delete

Identifier Design

URI Path Design

Rules

- A singular noun should be used for document names
- A plural noun should be used for collection names
- A verb or verb phrase should be used for controller names
- Variable path segments may be substituted with identity-based values
  http://api.library.restapi.org/section/{sectionId}/books/{bookId}/version/{versiond}
- CRUD function names should not be used in URIs

Identifier Design

- URI Query Design
- As a component of a URI, the query contributes to the unique identification of a resource.
- Rule: The query component of a URI may be used to filter collections
- Rule: The query component of a URI should be used to paginate collections
- **Represent relationships clearly in the URI**

Filter and Paginate

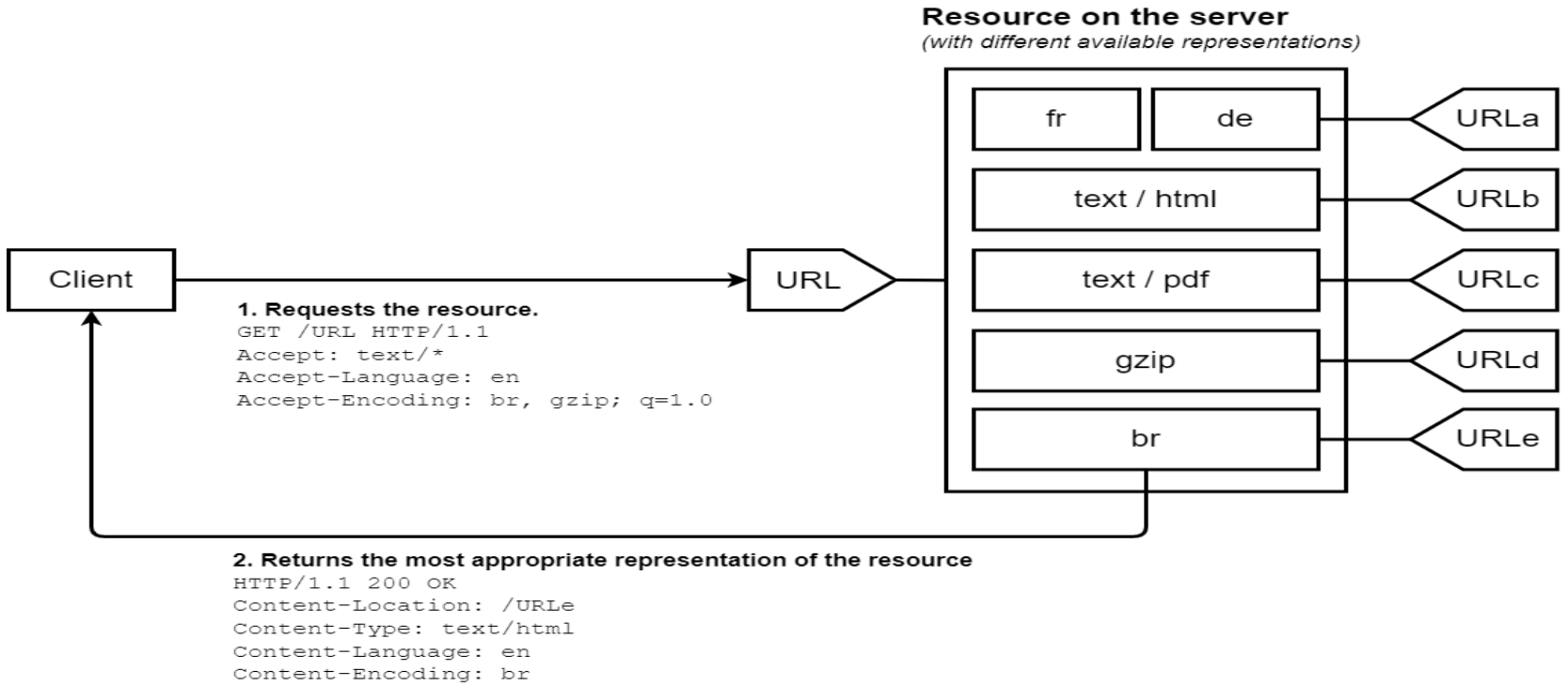
- */orders?minCost=n*
- /orders?limit=25&offset=50

Metadata Design

- Important HTTP headers
  - Content type
  - Content length
  - Last-modified
  - E-tag
  - Location

Metadata Design

- Media type negotiation should be supported when multiple representations are available



Representation Design

- Representation is the technical term for the data that is returned
- Users can view the representation of a resource in different formats, called media types
- JSON is the preferred format
- XML and other formats may optionally be used for resource representation
- A consistent form should be used to represent media types, links, and errors

Representation Design

- Keep the choice of media types and formats flexible to allow for varying application use cases, and client needs for each resource.
- Prefer to use well-known media types for representations.
- What data to include in JSON-formatted representations
- An example of a representation of a person resource:

```
{
"name" : "John",
"id" : "urn:example:user:1234",
"link" : {
"rel : "self",
"href" : "http://www.example.org/person/john"
},
"address" : {
"id" : "urn:example:address:4567",
"link" : {
"rel : "self",
"href" : "http://www.example.org/person/john/address"
}
```

Best Practices - Summary

- Use only nouns for a URI;
- GET methods should not alter the state of resource;
- Use plural nouns for a URI;
- Use sub-resources for relationships between resources;
- Use HTTP headers to specify input/output format;
- Provide users with filtering and paging for collections;
- Version the API;
- Provide proper HTTP status codes.

Best Practices

- Consistent API s - reduces the cognitive load on developers.
- Meaningful errors

  Example:

  Authentication failed because token is revoked : token_revoked or invalid_auth

  Value passed for name exceeded max: length name_too_long or invalid_name

API Design

- "*The API should enable developers to do one thing really well.It's not as easy as it sounds, and you want to be clear on what the API is not going to do as well.*"
    - *- Ido Green, developer advocate at Google*

RESTful Services Example - Customer

HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

Thank You!