

REST

- REST is an architectural style for APIs over HTTP
 - organized around *resources* accessed/modified using standard web requests (PUT, GET, POST, DELETE)
 - a resource is anything that can be named – stored data or the result of a computation or service
 - *representational state transfer*
 - client doesn't touch the resources held by the server directly, instead it sends and receives snapshots of their data
 - a set of conventions, not a library or a protocol
- REST properties – guiding constraints
 - *client/server* – clients are separated from servers by a well-defined interface
 - client only knows the URI of the requested resource
 - *stateless* – each request must include all necessary info for processing, server doesn't store info between requests
 - *uniform interface* – every resource is identified by a URI and acted on using the same small set of standard HTTP methods
 - *cache* – resources can be marked as cacheable to improve performance
 - *layered system* – client can't tell if it is connected directly to the server or if there are intermediaries

1

Resources, Methods, and Endpoints

Resource names are nouns –

- /books – all books (a collection)
- /books/42 – the book whose id is 42 (an item)
- /members/7/loans – member 7's loans (nested)
- /books?author=Adams – query parameter filters a collection

– convention rather than requirement
– collections are plural names
– items are identified by key
– nesting reflects FKs
– the query string says which subset

The HTTP method carries the action –

- POST /books – create
- GET /books/42 – read
- PUT /books/42 – update
- DELETE /books/42 – delete

anti-pattern: GET /deleteBook?id=42
– URL duplicates the method, making contradictions possible
– breaks safety and caching of GET – tools that discover the URL (crawlers, prefetchers) will trigger the delete
– destroys the uniform interface – each action needs a separate name
– hides the resource model – thinking in verbs means designing actions instead of resources
– key belongs in the path, not the query string

An *endpoint* is a specific operation that the API provides – an HTTP method plus a URL path.

2

JSON For Data Transfer

request

```
POST /books
Content-Type: application/json
```

} HTTP headers

```
{
  "isbn": "9780345391803",
  "title": "The Hitchhiker's Guide to the Galaxy",
  "author": "Douglas Adams"
}
```

} body

contains the data itself,
written as JSON

} key names don't need
to match the column
names

response

```
201 Created
Location: /books/42
```

} HTTP headers

```
{
  "bookId": 42,
  "isbn": "9780345391803",
  "title": "The Hitchhiker's Guide to the Galaxy",
  "author": "Douglas Adams",
  "available": true
}
```

} body

contains the data itself,
written as JSON

} key names don't need
to match the column
names

JSON – an object consists of { } surrounding a comma-separated list of key-value pairs
keys are strings
values can be text, numbers, true/false, null, a list, or another object

Status Codes

method	example	typical success
POST	POST /books	201 Created
GET	GET /books/42	200 OK
PUT	PUT /books/42	200 OK (or 204)
DELETE	DELETE /books/42	204 No Content

Status codes say how it went –

- first digit identifies if it worked and if not whose fault it was
- 2xx success
 - 200 OK – the request succeeded and the body holds the result
 - 201 Created – a new resource was created; the body holds the new resource
 - 204 No Content – the request succeeded and there's nothing to send back
- 4xx client's fault
 - 400 Bad Request – the request itself is wrong
 - 401 Unauthorized – authentication problem (the server doesn't know who you are)
 - 403 Forbidden – authorization problem (the server knows who you are, but you aren't allowed to do the action)
 - 404 Not Found – the resource doesn't exist or the client isn't allowed to know about it
 - 409 Conflict – request is valid but clashes with the current state of the data (violates a constraint)
- 5xx server's fault: 500 Internal Server Error

Library Demo

MEMBERS (member_id PK, username UNIQUE, password_hash, role ENUM('member','librarian'))

BOOKS (book_id PK, isbn UNIQUE, title, author, published_on DATE NULL, replacement_cost DECIMAL(6,2) NULL, available BOOLEAN NOT NULL DEFAULT TRUE)

LOANS (loan_id PK, book_id FK -> BOOKS ON DELETE RESTRICT, member_id FK -> MEMBERS ON DELETE CASCADE, checked_out_on DATE, due_on DATE, returned_on DATE NULL)

- loans connect a member and a book

use case	endpoint	CRUD	SQL	success	failure cases
browse all books					
find books by an author					
add a new book					
view a book's details					
edit a book's details					
remove a book					
list loans					
view one loan					
check out a book					
return a book					

use case	endpoint	CRUD	SQL	success	failure cases
browse all books	GET /books	read	SELECT ... FROM BOOKS	200 OK	none yet
find books by an author	GET /books?author=...	read	SELECT ... WHERE author = ?	200 OK	none (no match = empty list)
add a new book	POST /books	create	INSERT INTO BOOKS ...	201 Created	400 (title missing)
view a book's details	GET /books/{id}	read	SELECT ... WHERE book_id = ?	200 OK	404 (no such book)
edit a book's details	PUT /books/{id}	update	UPDATE BOOKS ... WHERE book_id = ?	200 OK	400 (new data is invalid), 404 (no such book)
remove a book	DELETE /books/{id}	delete	DELETE FROM BOOKS WHERE book_id = ?	204 No Content	404 (no such book), 409 (deleting a book with loans)
list loans	GET /loans	read	SELECT ... FROM LOANS	200 OK	none yet
view one loan	GET /loans/{id}	read	SELECT ... WHERE loan_id = ?	200 OK	404 (no such loan)
check out a book	POST /loans	read, create, update	SELECT ... FROM BOOKS WHERE book_id = ? INSERT INTO LOANS ... UPDATE BOOKS SET available=FALSE WHERE book_id = ?	201 Created	400 (missing book id), 404 (no such book), 409 (already checked out)
return a book					

Returning a Book

- “return a book” doesn't map neatly to one CRUD verb – it is an event with rules, not just an update
 - the client doesn't choose the rules or updated values – the server sets returned_on to today
 - it is a one-way state change – a loan goes from out to returned, once
 - it involves more than one resource – updates the loan and sets BOOKS.available

Returning a Book

- option – treat the return as an edit
 - PUT `/loans/{id}` with a return date
 - body of the PUT can contain a full loan (all fields) – server must check everything other than `returnedOn` to make sure there are no changes
 - PATCH `/loans/{id}` with only `returnedOn` still treats the return as an edit but limits the change to one field
- option – treat “a return” as a resource
 - POST `/loans/{id}/return` creates a return
 - no body – all info is in the `{id}` value
- no single right answer
 - as a resource is clearer about intent (names the event) but `/return` is a noun but not really a thing – no row or table backs it, and special actions pile up
 - as an edit reflects that a return is really just an update of a loan, but that reveals implementation details

Library Demo

use case	endpoint	CRUD	SQL	success	failure cases
return a book	PUT <code>/loans/{id}</code>	read, update	UPDATE LOANS SET <code>returned_on = ?</code> ... UPDATE BOOKS SET <code>available = TRUE</code> ...	200 OK	400 (other fields changed), 404 (no such id), 409 (already returned)
return a book	PATCH <code>/loans/{id}</code>	read, update	UPDATE LOANS SET <code>returned_on = ?</code> ... UPDATE BOOKS SET <code>available = TRUE</code> ...	200 OK	400 (bad date), 404 (no such id), 409 (already returned)
return a book	POST <code>/loans/{id}/return</code>	read, update	UPDATE LOANS SET <code>returned_on = CURRENT_DATE</code> ... UPDATE BOOKS SET <code>available = TRUE</code> ...	204 No Content	404 (no such id), 409 (already returned)