REST

CODE FELLOWS

CODE FELLOWS

401
Advanced
Javascript

REST

REpresentational **S**tate **T**ransfer

Guiding Principles

1. **Client–server** – Separate the user interface concerns from the data storage concerns
2. **Stateless** – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server.
3. **Cacheable** – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable (client can cache)
4. **Uniform interface** – REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.
5. **Layered system** – Layered systems must constrain component behavior such that each component cannot “see” beyond the immediate layer with which they are interacting.
6. **Code on demand (optional)** – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts.

401
Advanced
Javascript

The Resource

The key abstraction of information in REST is a resource.

Any information that can be named can be a resource: a document or image, a collection, database record

REST uses a **resource identifier** to identify the resource involved in an interaction between components.

The state of resource at any particular timestamp is known as resource representation.
Data, metadata describing state, links to help the client transition to the next state, etc.

A truly RESTful API looks like *hypertext*. Every addressable unit of information carries an address, either explicitly (e.g., link and id attributes) or implicitly (e.g., derived from the media type definition and representation structure).

Resource representations must be self-descriptive: the client does not need to know if a resource is employee or device.

401
Advanced
Javascript

Resource Methods (verbs)

GET	Retrieves a resource (read only)
HEAD	Asks for a response identical to that of a GET request, but without the response body.
POST	Sends “unlimited” data to the resource.
PUT	Requests that the enclosed entity be stored under the supplied URI
DELETE	The DELETE method deletes the specified resource.
TRACE	The TRACE method echoes the received request
OPTIONS	The OPTIONS method returns the HTTP methods that the server supports for the URI
CONNECT	The CONNECT method converts the request connection to a transparent tunnel
PATCH	The PATCH method applies partial modifications to a resource

https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

CODE FELLOWS

401
Advanced
Javascript

The URI

scheme:[**//**[**user**[**:password**]**@**]**host**[**:port**]][**/path**][?**query**][#**fragment**]

Scheme: http, ftp, gopher, file, mailto, etc. This identifies the type/protocol

User: Optional Username
Password: Optional Password

These are mainly used in SSH or secure connection URIs

Host: The server name you are attempting to connect to
Port: A specific port on that server (80 is the default for web, and can be omitted)

Resource: A path to the specific resource you are requesting

Query: Parameters to be sent into the resource (?this=that&foo=bar)

Fragment: Generally used to identify a part on a page, but has other uses in Client Side JS

https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

401
Advanced
Javascript

So, what's the point?

REST uses HTTP and URIs to give us a way of managing state in a controlled manner. It's just another type of conversation...

- Go get me this thing
- I'll use that thing to do some stuff
- I might use that thing to fetch a new thing
- I might mess with that thing
- If I need to save that thing, I'm going to send it back with an instruction
- You can work your magic on it
- And give it back

401
Advanced
Javascript

CODE!

Lets create a simple CRUD application using RESTful Verbs

Model: Note

CREATE:	POST	/api/notes
READ ALL:	GET	/api/notes
READ:	GET	/api/notes/?id=[id]
UPDATE:	PUT	/api/notes/?id=[id]
DESTROY:	DELETE	/api/notes?id=[id]