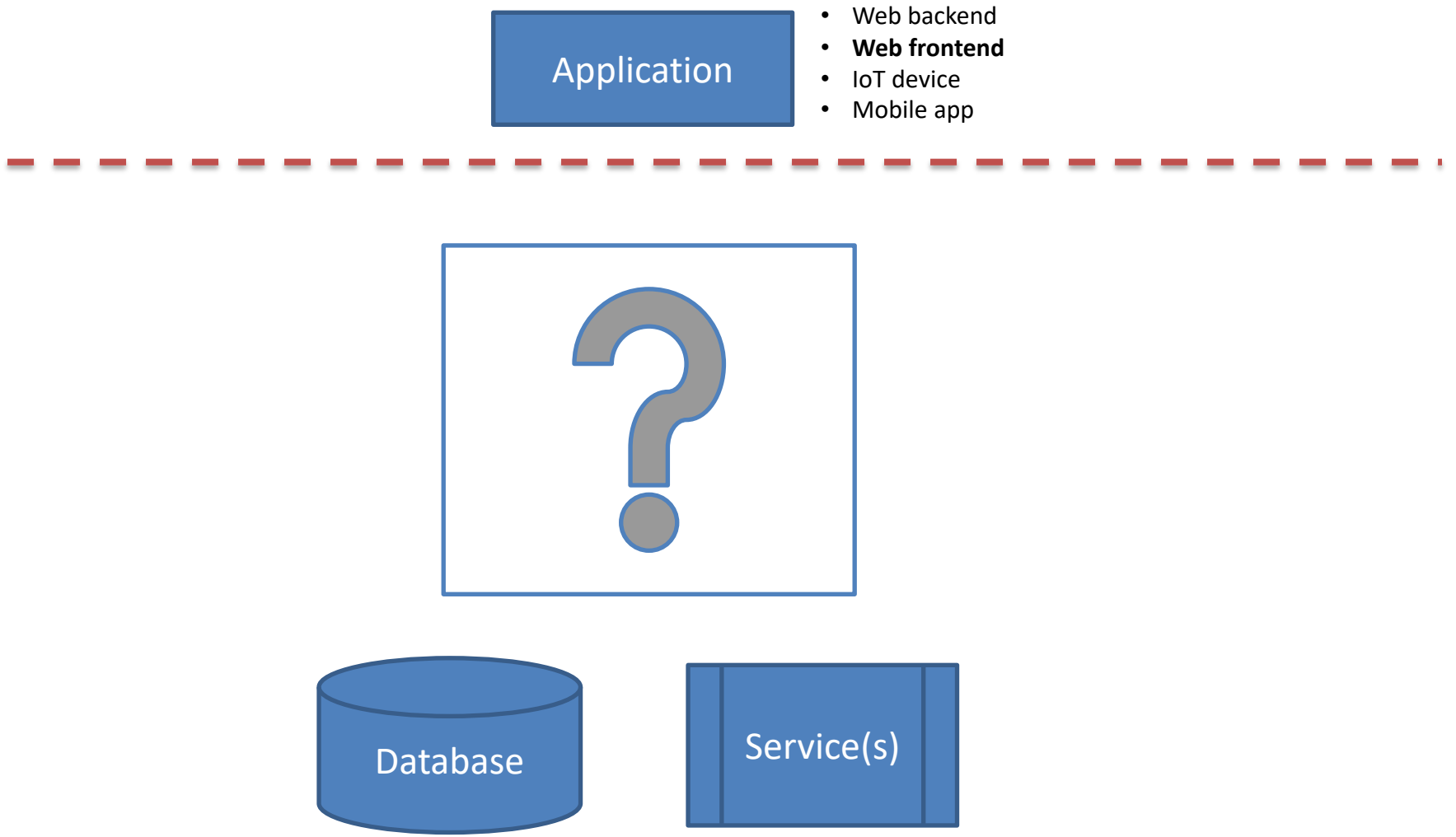
<WA1/>
<AW1/>
2024

HTTP APIs

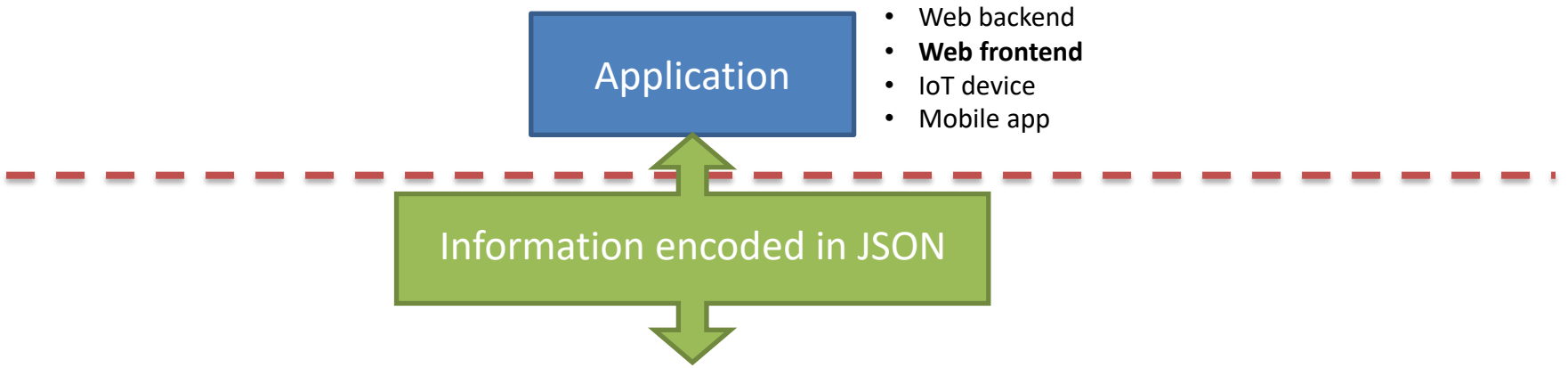
The glue between clients and servers

Fulvio Corno
Luigi De Russis

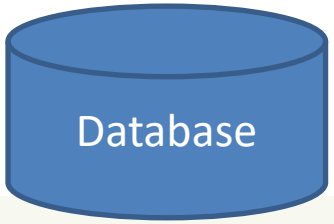
Goal


Application
Web backend
Web frontend
IoT device
Mobile app
Database
Service(s)

Architecture



- Web backend
- **Web frontend**
- IoT device
- Mobile app



JSON - JavaScript Object Notation

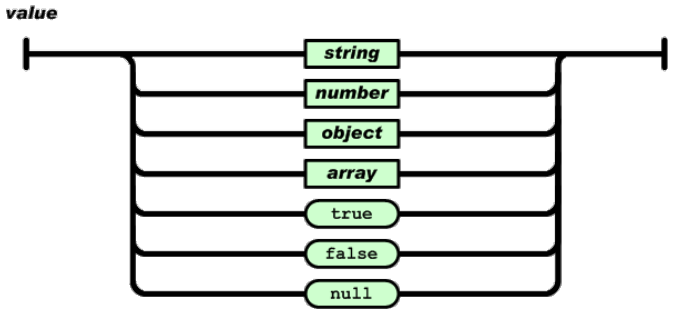
/ˈdʒeɪ·sən/

- Lightweight Data Interchange Format
  - Subset of JavaScript syntax for object literals
  - Easy for humans to read and write
  - Easy for machines to parse and generate
  - https://www.json.org/
  - ECMA 404 Standard: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf
  - RFC 8259: https://tools.ietf.org/html/rfc8259
- Media type: `application/json`

JSON è un formato di scambio di dati
sottoinsieme della sintassi di JavaScript
con l'obbiettivo che sia facile sia da leggere che s
scrivere che per le macchine da parsificare
standard EGMA, standard EIPF
Apllication Jeson

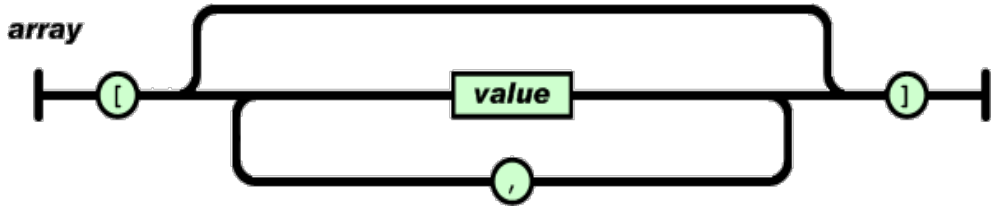
JSON Logical Structure

- Primitive types: string, number, true/false/null
  - Strings MUST use "double" quotes, not 'single'



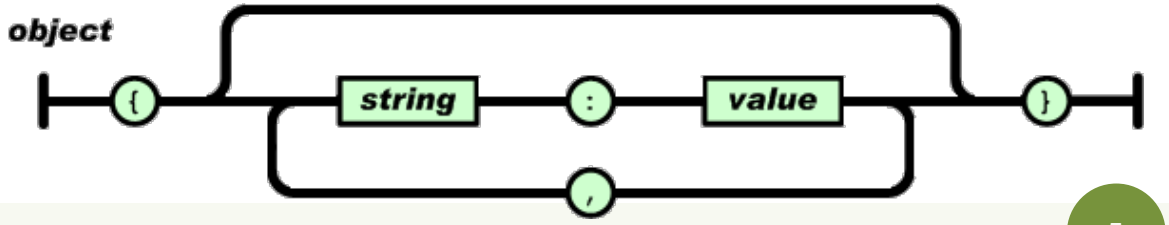
- Composite type – Array: ordered lists of values

[...]



- Composite type – Objects: list of key-value pairs
  - Keys are strings (not identifiers)
  - MUST be "quoted"

{...}



JSON Example


```
{
    "firstName": "John",
    "lastName": "Smith",
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        "212 555-1234",
        "646 555-4567"
    ]
}
```

Name/Value Pairs

Child properties

String Array

Number data type

Using JSON in JavaScript

prende un oggetto di JavaScript e lo converte in stringa di JSON

- `JSON.stringify` to convert objects into JSON
  - `const jsonString = JSON.stringify(myObj)`
  - Works recursively also on nested objects/arrays
  - Excludes function properties (methods) and undefined-valued properties

undefine -> non se lo porta in JSON

operazione opposta, prende una stringa di JSON e cerca di restituire un oggetto JavaScript che corrisponde alla stringa

- `JSON.parse` to convert JSON back into an object
  - `const myObj = JSON.parse(jsonString)`
  - All created objects have the default {} Object prototype
    - Can fix with a *reviver* callback

https://javascript.info/json

Main Types of URIs

- **Collection** URI
  - Represents a set (or list) of objects (or items) of the same type
  - Format: /collection
    - `http://api.polito.it/students`
    - `http://api.polito.it/courses`

- **Element** (Item, Simple) URI
  - Represents a single item, and its properties
  - Format: /collection/identifier
    - `http://api.polito.it/students/s123456`
    - `http://api.polito.it/courses/01zqp`

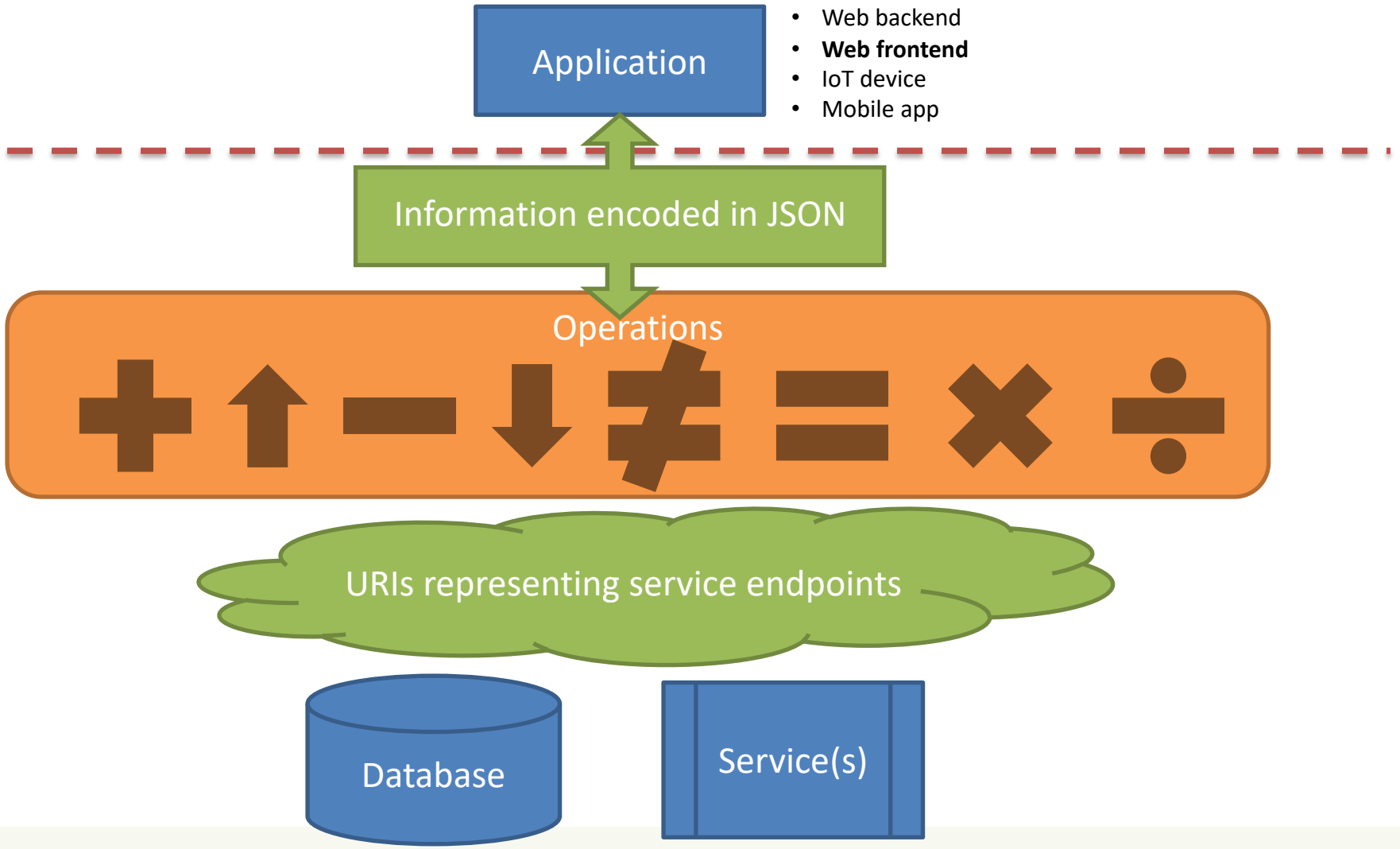
Best Practice for URIs Definition

- Nouns (not verbs)
- Plural nouns
- Concrete names (not abstract)
  - /courses, not /items

Operations on Server URIs

- The server supports operations on the specified object/collection(s)
  - Add
  - Delete
  - Update
  - Find
  - Search
  - …

Architecture



Actions Use HTTP Methods

- GET
  - Retrieve the representation of the resource (in the HTTP response body)
  - Collection: the list of items
  - Element: the properties of the element
- POST
  - Create a new resource (data in the HTTP request body)
  - Use a URI for a Collection
- PUT
  - Update an existing element (data in the HTTP request body)
  - Mainly for elements' properties
- DELETE

Actions on Resources

Resource	GET	POST	PUT	DELETE
Collection	Retrieve the list of items	Add a new element to the collection	-	-
Single Element	Retrieve the properties of the element	-	Replace the values of the element properties	Delete the element

Actions on Resources: Example

Resource	GET	POST	PUT	DELETE
/dogs	List dogs	Create a new dog	Bulk update dogs (avoid)	Delete all dogs (avoid)
/dogs/1234	Show info about the dog with id 1234	ERROR	If exists, update the info about dog #1234	Delete the dog #1234

Standard Methods

Standard Method	HTTP Mapping	HTTP Request Body	HTTP Response Body
List	GET <collection URL>	N/A	Resource* list
Get	GET <resource URL>	N/A	Resource*
Create	POST <collection URL>	Resource	Resource*
Update	PUT or PATCH <resource URL>	Resource	Resource*
Delete	DELETE <resource URL>	N/A	google.protobuf.Empty**

https://cloud.google.com/apis/design/standard_methods

Relationships

- A given Element may have a (1:1 or 1:N) relationship with other Element(s)
- Represent with: /collection/identifier/relationship
- `http://api.polito.it/students/s123456/courses` (list of courses followed by student `s123456`)
- `http://api.polito.it/courses/01qzp/students` (list of students enrolled in course `01qzp`)

Complex Resource Search

- Use `?`*`parameter`*`=`*`value`* for more advanced resource filtering (or search)
  - E.g., `https://api.twitter.com/1.1/statuses/user_timeline.json`**`?screen_name=twitterapi&count=2`**

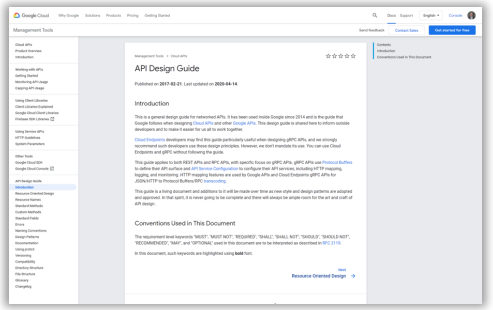
Errors

- When errors or exceptions are encountered, use meaningful HTTP Status Codes
  - The Response Body may contain additional information (e.g., informational error messages)

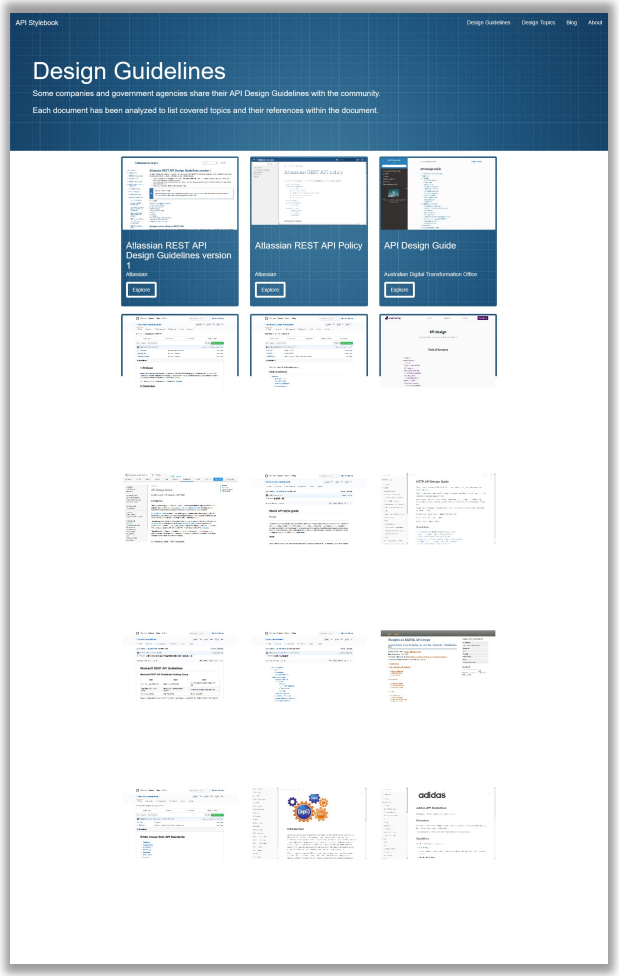
```
{
   "developerMessage" : "Verbose, plain language description of
the problem for the app developer with hints about how to fix
it.",
   "userMessage":"Pass this message on to the app user if
needed.",
   "errorCode" : 12345,
   "more info": "http://dev.teachdogrest.com/errors/12345"
}
```

API Design

- How to design a set of APIs for your application?
- Practical guidelines, with applied standard practices
- Suggestion: Google API Design Guide
  - https://cloud.google.com/apis/design/



http://apistylebook.com/design/guidelines/



Guidelines for implementing back-end APIs

HTTP APIS IN EXPRESS

HTTP APIs implementation

- HTTP API endpoints are just regular HTTP requests
- Request URL contains the Element Identifiers (`/dogs/1234`)
  - Extensive usage of parametric paths (`/dogs/:dogId`)
- Request/response Body contains the Element Representation (in JSON)
  - **Request:** `req.body` populated by the `express.json()` middleware
  - **Response:** `res.json()` to send the response
- Always validate input parameters
- Always validate input parameters
- Really, always validate input parameters

Collections

Elements

GET

```
app.get('/answers', (req, res) => {
  dao.listAnswers().then((answers) => {
    res.json(answers);
  });
});
```

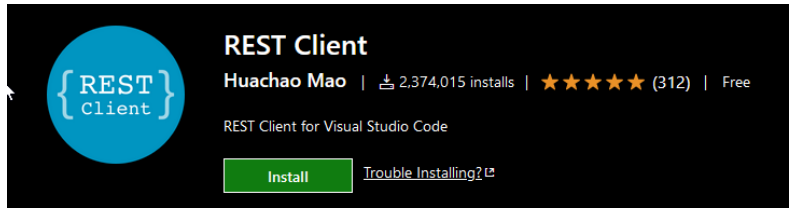
```
app.get('/answers/:id', (req, res) => {
  // TODO: validation of req.params.id
  dao.readAnswer(req.params.id)
  .then((answer)=>res.json(answer));
});
```

POST

```
app.use(express.json());

app.post('/answers', (req, res) => {
  const answer = req.body;
  // TODO: validation of answer
  dao.createAnswer(answer);
});
```

Testing HTTP APIs



https://marketplace.visualstudio.com/items?itemName=humao.rest-client

- You may use the "REST Client" extension for VSCode
- Create a file with extension `.http`
- Write one or more HTTP *Requests* (separated by ###)
  - Method + URL
  - Request headers (optional)
  - Request body (optional, after empty line)
- Click on the 'Send Request' link that will appear
  - A new Tab will open, with the *Response* headers and body

```
GET https://example.com/comments/1 HTTP/1.1

###

GET https://example.com/topics/1 HTTP/1.1

###

POST https://example.com/comments HTTP/1.1
content-type: application/json

{
    "name": "sample",
    "time": "Wed, 21 Oct 2015 18:27:50 GMT"
}
```

License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
  - **Share** — copy and redistribute the material in any medium or format
  - **Adapt** — remix, transform, and build upon the material
  - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
  - **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - **NonCommercial** — You may not use the material for commercial purposes.
  - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- https://creativecommons.org/licenses/by-nc-sa/4.0/