ROASTER
router

TO BUILD ON

TO BUILD WITH

3/77 – Photo by ? (edit) https://unsplash.com/

ROUTING IS MAPPING AN HTTP REQUEST TO AN OPERATION

ROUTING IS MAPPING AN HTTP REQUEST TO A *Function*

```
{
   "openapi": "3.0.0",
   "info": {
       "version": "1.0.0",
       "title": "My API",
       "description": "Play a game"
   },
   "servers": { ""}
   "paths": {}
}
```

OPEN API SPEC V3.0.1

- vendor neutral
- existing tooling

ROUTE

/THINGS/{ID}

```
{
  "/things/{id}": {
    "get": {
      "description": "Get the thing",
      "operationId": "thing:detail-view",
      "parameters": {
        "id": {
          "in": "path",
          "type": "string"
}}}}}
```

REQUEST

/THINGS/42

```
map {
  "id": "1f4edcdf-b4e0-4e69-8edc-dfb4e09e6937",
  "body": (),
  "parameters": { "id": "42" },
  "user": map {
    "name": "guest",
    "groups": [ "guest" ],
    ...
  }
  ...
}
```

```
module namespace thing="my/thing";

declare function thing:detail-view($request) {
  thing:by-id($request?parameters?id)
};

declare function thing:by-id($id as xs:string) {
  collection('/db/my-data/things')/id($id)
};
```

$REQUEST

?PARAMETERS

- **Where is it expected?**
  - `path`, `query`, `cookie`, `header`
- **Which type does it have?**
  - `object`, `string`, `boolean`, `number`, `date`, **…**
- **Which format does it have?**
  - `password`, `binary`, `base64`

$REQUEST

?BODY

15/77 – e-editiones community 2022

Only parsed on

`post`, `put`, `patch`

`get` will never have a body

```
{
  "post": {
    "operationId": "auth:login",
    "requestBody": {
      "required": true,
      "content": {}
}}}
```

```
{
  "multipart/form-data": {
    "schema": {
      "properties": {
        "user": { "type": "string" },
        "password": {
          "type": "string",
          "format": "password"
}}}}}
```

RESPONSE

```
{
  "/things/{id}": {
    "get": {
      "description": "Get the thing",
      "operationId": "thing:detail-view",
      "parameters": {
        "id": { "in": "path", "type": "string" }
      }
}}}
```

```
{
  "/things/{id}": {
    "get": {
      "description": "Get the thing",
      "operationId": "thing:detail-view",
      "parameters": {
        "id": { "in": "path", "type": "string" }
      },
      "responses": {}
}}}
```

```
{
  "200": {
    "description": "Access Token valid",
    "content": {
       "application/xml": {
         "schema": {
           "type": "object"
  }}}},
  "404": {},
  "401": {}
}
```

```
(:~
 : random number determined
 : by a regular throw of a dice
 :)
declare function my:random ($request) {
  4
};
```

```
(: Make a scene :)
declare function my:drama ($request) {
  error(
    $errors:BAD_REQUEST,
    "I CAN'T work like that",
    $request (: pass additional data :)
  )
};
```

```
(: artisanal, custom-made response :)
declare function my:upload ($request) {
  roaster:response(
    201,
    xmldb:store(
      "/db/apps/roasted/uploads",
      $request?parameters?path,
      $request?body
    )
  )
};
```

```
(: look elsewhere :)
declare function my:redirect ($request) {
  roaster:response(
    301,
    "text/plain",
    "redirecting",
    map { "Location": "?ping" }
  )
};
```

```
(: look elsewhere :)
declare function my:redirect ($request) {
  roaster:response(
    301,                                (: CODE :)
    "text/plain",                       (: TYPE :)
    "redirecting",                      (: BODY :)
    map { "Location": "?ping" } (: HEADERS :)
  )
};
```

CODE
TYPE
BODY
HEADERS

```
router:RESPONSE_CODE
router:RESPONSE_TYPE
router:RESPONSE_BODY
router:RESPONSE_HEADERS
```

```
map {
                router:RESPONSE_CODE    : xs:integer,
                router:RESPONSE_TYPE    : xs:string,
                router:RESPONSE_BODY    : item()*,
                router:RESPONSE_HEADERS : map(xs:string, xs:string)
}
```

```
declare variable $router:RESPONSE_CODE    := xs:QName("router:RESPONSE_CODE");
declare variable $router:RESPONSE_TYPE    := xs:QName("router:RESPONSE_TYPE");
declare variable $router:RESPONSE_HEADERS := xs:QName("router:RESPONSE_HEADERS");
declare variable $router:RESPONSE_BODY    := xs:QName("router:RESPONSE_BODY");
```

MEDIA TYPES

Does this route allow this media to be sent?

```
{
  "content": {
    "text/plain": {
      "schema": {
        "type": "string" }
}}}
```

Determine serialization method from declared media type.

xml

- `application/xml`
- `text/xml`

json

- `application/json`

html5

- `text/html`

xhtml

- `application/xml+xhtml`

text

- `image/jpeg`
- `application/javascript`
- `application/octet-stream`

xml

- `application/tei+xml`
- `application/*+xml`
- `image/svg+xml`

json

- `application/ld+json`
- `application/*+json`

anything else is text

- */*

FILE UPLOAD

PREVIOUSLY

- handle only single file uploads
- filename has to be provided separately

```
{
  "operationId": "upload:body",
  "parameters": {
    "path": { "in": "path", "type":"string" }},
  "requestBody": {
    "content": {
      "*/*": {
        "schema": {
          "type": "string",
          "format": "binary"
}}}}}
```

```
declare function upload:body ($request) {
  roaster:response(
    201,
    xmldb:store(
      $upload:collection,
      $request?parameters?path,
      $request?body
    )
  )
};
```

NOW

- proper handling of array properties in form-data
- automatic detection and handling of multiple bodies
- thus, able to handle batch uploads

```
{
  "operationId": "upload:batch",
  "requestBody": {
    "content": {
      "multipart/form-data": {
        "schema": {
          "type": "object",
          "properties": {
            "file": {
              "type": "array",
              "items": {
                "type": "string",
                "format": "binary"
}}}}}}}}
```

```
declare function upload:batch ($request as map(*)) {
    try {
        let $download-links as xs:string+ :=
            for $file in $request?body?file
            let $stored :=
                xmldb:store($upload:collection, $file?name, $file?data)
            return $upload:download-path || $file?name

        return
            roaster:response(201,
                map{ "uploaded": array{ $download-links }})
    }
    catch * {
        roaster:response(400, map { "error": $err:description })
    }
};
```

ERROR HANDLING

- central handling of errors
- custom error handlers

```
{
  "get": {
    "x-error-handler": "my:html-error"
  }
}
```

```
declare function my:html-error(
  $error as map(*)
) as element(html) {
  <html><body>
    <h1>Ah Snap!</h1>
    <p>{$error?description}</p>
    <code>{$error?code}</code>
  </body></html>
};
```

AUTHENTICATION

53/77 – Photo by Tim Mossholder https://unsplash.com/s/photos/road-sign

```
{
  "securitySchemes": {
    "basicAuth": {
      "type": "http",
      "scheme": "basic"
    },
    "cookieAuth": {
      "type": "apiKey",
      "name": "roasted.com.login",
      "in": "cookie"
}}}
```

```
{
  "security": [ "cookieAuth", "basicAuth" ]
}
```

AUTHORISATION

Allow only members of the physics group

```
{
  "x-constraints": { "groups": "physics" }
}
```

Allow members of both the physics and dba group

```
{
  "x-constraints": { "groups": ["physics", "dba"] }
}
```

Allow only a specific user

```
{
  "x-constraints": { "user": "admin" }
}
```

LOGGING

LOGGING

- each request and each response are logged
  - will always include the request id

```
2021-11-01 12:34:56,370 ... request post /login
2021-11-01 12:34:56,388 ... post /login: 200
2021-11-01 12:34:56,394 ... request get /api/parameters
2021-11-01 12:34:56,400 ... get /api/parameters: 200
2021-11-01 12:34:56,431 ... request get /logout
2021-11-01 12:34:56,437 ... get /logout: 301
2021-11-01 12:34:56,455 ... request get /logout
2021-11-01 12:34:56,458 ... get /logout: 200
```

```
... [...351ab6cf7fb8] request post /login
... [...351ab6cf7fb8] post /login: 200
... [...56f2b155c571] request get /api/parameters
... [...56f2b155c571] get /api/parameters: 200
... [...645dd65dff94] request get /logout
... [...645dd65dff94] get /logout: 301
... [...715c59e53396] request get /logout
... [...715c59e53396] get /logout: 200
```

```
... INFO [...ddf283193aee] request post /api/errors/handle
... ERROR [...ddf283193aee] get /api/errors/handle: 500
{
    "code": "err:XPDY0002",
    "description": "[...] variable '$undefined' is not set.",
    "value": null,
    "module": "String/-23433160863015345 43",
    "line":42, "column":5
}
```

MIDDLEWARE

- receives both request and response data
  - is allowed to modify *both*
    - widely adopted concept

```
function(map(*), map(*)) as map(*)+
```

```
declare function my:beep-boop (
    $request as map(*), $response as map(*)
) as map(*)+ {
    (: modify request :)
    map:put($request, "beep", "boop"),
    (: modify response :)
    map:put($response, $router:RESPONSE_HEADERS,
        map:merge((
            $response?($router:RESPONSE_HEADERS),
            map { "x-beep" : "boop" }
        ))
    )
};
```

CUSTOM AUTHORIZATION

```
{
    "JWTAuth": {
        "type": "apiKey",
        "in": "header",
        "name": "X-Auth-Token"
    }
}
```

```
{
  "security": [ "cookieAuth", "JWTAuth" ]
}
```


TODO
frei

OPENAPI SPEC

- References with `$ref`
  - Schema Validation
- also look at the open issues on github

TOOLING

- app template

... AND MORE DOCUMENTATION

- roaster repository
- Readme
- Tests
- Example Application roasted
- OpenAPI Documentation

 – Photo by Matt Botsford https://unsplash.com/s/photos/road-sign

All this is coming to your

TEI-PUBLISHER V8

 – Photo by Krzysztof Hepner on Unsplash

JURI LEINO

@LINE-0

@_0@CHAOS.SOCIAL

77/77 – Photo by Jonathan Kemper https://unsplash.com/s/photos/road-sign