My Rules of REST

These match most common REST implementations

- URL represents a "resource" to interact with
- HTTP method is the interaction with the resource
- HTTP Status code is interaction result

First Rule of REST

First Rule of REST:

- The URL represents a "resource" to interact with

Often a noun (the HTTP method is the verb)

- **Good** - `/student/`
- **Good** - `/grades/`
- **Good** - `/locations/`
- **Bad** - `/addStudent/`
- **Bad** - `/updateGrade/`
- **Bad** - `/searchLocations/`

URL as resource

- Parameters: in query, body, or path
- Often different based on method
  - `GET /students`
  - `GET /students?startsWith=Am`
  - `POST /students?givenName=Xiu&familyName=Li`
  - `POST /students/Li/Xui/`
  - `PATCH /stduents/34322/`
  - `DELETE /students?billingStatus=overdue`
- **The path of the URL identifies the "thing"**
  - Params do NOT identify the "thing" (resource)

Second Rule of REST

- HTTP method is the interaction with the resource

The URL is the "thing"

The method is what you "do" to it

Examples of the Second Rule of REST

The method shows the kind of interaction:

- `GET /students/` - read
- `POST /students/` - create
- `PUT /students/Naresh/Rajkumar` - overwrite
- `DELETE /students/Naresh/Rajkumar` - remove
- `PATCH /students/Naresh/Rajkumar` - partial update

These have passed params, but

- Method and the URL alone say what is happening

POST vs PUT vs PATCH

Common confusion: Create vs overwrite vs update

- POST (create)
    - No existing record
- PUT (replace)
    - Replace existing record
    - Save nothing from it
- PATCH (update)
    - Replace certain fields in the record
    - Unmentioned fields stay as-is

What is passed/received?

- `POST /students/` - **create**
  - Send: (data for new student)
  - Get: (url or data to identify new record)
- `PUT /students/Naresh/Rajkumar` - **overwrite**
  - Send: (data to replace with)
  - Get: ?
- `PATCH /students/Naresh/Rajkumar` - **partial update**
  - Send: (fields with changed values)
  - Get: (? most often updated record)

Third Rule of REST

- HTTP Status code is interaction result

There are many Status codes!

- With meaningful names
- Use them!
- but confirm the meaning (MDN)

Add details in body

Status Codes

Some general "classes" of status codes

- 100-199 (1xx): Informational (very rare)
- 200-299 (**2xx**): Successful
- 300-399 (**3xx**): Redirection
- 400-499 (**4xx**): Error (client-caused)
- 500-599 (**5xx**): Error (server-side)

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

REST Status Code Examples

Some common scenarios

- **200 (OK)** - Means real success
- **400 (Bad Request)** - bad input
    - Provide detail in body of response
- **404 (Not Found)**
- **500 (Internal Server Error)** - server had issue
    - Not user's fault
    - Not expected!

REST Response

- Other than HTTP Status code
  - Not much direction given
- Common responses (Can vary!)
  - If server created a UUID/ID for new resource
    - Provide in response
      - Record or URL
  - If a record changed
    - Provide the new record
  - If an error code
    - Provide details in body
    - Details in same format as success

JSON is common

JSON is common, even from non-JS services

Pro:

- Very portable
- Very readable

Con:

- No built-in schema validation
- No comments

Basic REST Express Example

```
const cats = {};

app.get('/cats', (req, res) => {
  res.json(Object.keys(cats));
});

app.get('/cats/:name', (req, res) => {
  const name = req.params.name;
  if(cats[name]) {
    res.json(cats[name]);
    return;
  }
  res.status(404).json({ error: `Unknown cat: ${name}` });
});
```

- `:name` syntax (express) sets the `req.params.name`
  - example: `GET /cats/Jorts`
- `.json()` does `JSON.stringify()`
  - AND sets the response `content-type` header

More REST Express Example

```
app.post('/cats', express.json(), (req, res) => {
  const name = req.body.name;
  if(!name) {
    res.status(400).json({ error: "'name' required" });
  } else if(cats[name]) {
    res.status(409).json({ error: `duplicate: ${name}`});
  } else {
    cats[name] = req.body;
    res.sendStatus(200);
  }
});
```

`express.json()` middleware requires request `content-type` of `application/json`, populates `req.body`

No `content-type` === no `body` value.

A REST service in express()

- Have a route URL that matches Rule 1
- Use a method that matches Rule 2
- Use correct status codes for Rule 3
- Parse incoming body data
  - `express.json()` for JSON
- Send JSON data in response
  - `res.json()`
- No HTML, No Redirects