RULES OF REST

These are my personal rules, but they will 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 can be query params or in body
- Can also be in url directly
- These are often different based on method
    - `GET /students`
    - `GET /students?startsWith=Am`
    - `POST /students?givenName=Xiu&familyName=Li`
    - `POST /students/Li/Xui/`
    - `PATCH /students/34322/`
    - `DELETE /students?billingStatus=overdue`
- BUT the path of the URL identifies the "thing", not the params

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 how you interact with the URL:

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

Most will pass params, but with just URL and method:

- What is happening
- to which resource

THIRD RULE OF REST

- HTTP Status code is interaction result

There are many Status codes!

- With meaningful names
- Use them!

Add any body to give details

REST STATUS CODE EXAMPLES

Some common scenarios

- **200 (OK)** - Only when the interaction worked
- **400 (Bad Request)** - bad input
  - Provide detail in body of response
- **404 (Not Found)**
  - Can be confusing - was the URL wrong, or was the resource just not there?
  - An error body can clear up the confusion
- **500 (Internal Server Error)** - server had issue
  - Not user's fault
  - Not expected!

REST RESPONSE BODY

- Services shouldn't give error messages for display
  - That moves UI changes to services (yuck)
  - Instead give error **codes** that are translated by client code
- JSON is common, even from non-JS services
  - Upside: very portable, very readable
  - Downside: No built-in schema validation

BASIC REST EXAMPLE

```
const people = {};

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

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

- `:name` syntax (express) sets the `req.params.name`
- `.json()` does `JSON.stringify()` AND sets the `content-type` header

MORE REST EXAMPLE

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

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

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

CONSIDERATIONS

- JSON for error messages?
- POST data needs to return new identifier
    - `POST /people/` - what url for new person?
- Long running requests need a polling setup
    - A slow query will timeout
    - Return a url to check
    - That "check" url gives result or not yet
- Versioning of services!
    - `/v1/people`
- path to services might conflict with pages
    - `/api/v1/people`

WRITE A REST SERVICE TO TRACK PEOPLE

- **GET** /people - JSON array of names
- **POST** /people/:name - Adds name, returns array
  - Status 409 (Conflict), `{error: "duplicate"}`
  - 400 (Bad Request), `{error: "missing-name"}`
- **DELETE** /people/:name - removes, returns array
  - 400 (Bad Request), `{error: "missing-name"}`

Consider:

- Are you looping through an array many times?
- Why these HTTP methods/verbs?
- Why return the array for each?

THINKING AHEAD

How would you add authorization requirements?

- pass a parameter that the service checks
- have a cookie that the service checks
- pass a special header that the service checks

What kinds of responses can this add?

- 401 - Authorization required
    - the thing to check wasn't there
- 403 - Forbidden
    - it was there but didn't allow access

SAMPLE AUTHENTICATION ENDPOINT

- **POST** `/api/v1/session` - sets a cookie ("logged in")
- **GET** `/api/v1/session` - client can check if logged in
- **DELETE** `/api/v1/session` - clears cookie ("logged out")
- **GET** `/api/v1/people`
    - Requires the cookie be set
    - ...with a value the server knows is valid
    - Returns a 401 value if cookie not set
    - Returns a 403 value if cookie has an invalid value
    - Other `/api/v1/(etc)` endpoints also do these checks and returns