My Rules of REST

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

These are **my summary** of the core REST concepts

- Can't Google "three rules of REST" and find this
 - But you'll find these work well as basics
- REST covers a lot, this is the **core**
 - Many "RESTful" implementations aren't pure
 - These "rules" are most commonly followed

First Rule of REST

The URL path represents a **resource** to interact with

- Path identifies the resource
- Often a noun (HTTP method is the verb)
- Plural if a collection
 - Good /students/
 - Good /grades/
 - Good /locations/
 - **Bad** /addStudent/
 - **Bad** /update-grade/
 - Bad /searchLocations/

What is a "Resource"?

"Resource" is a bit abstract (intentionally)

- A record?
- A concept?

Separates caller of service from actual data

• Allows for changes in actual data structure

Resource is some concept you can interact with

- Students
- Grades
- Semesters

More REST Rule 1: URL Path as resource

- Path identifies the individual resource
 - Query Params may filter results
- Multiple methods may use same Path
 - GET /students
 - GET /students?startsWith=Am
 - POST /students
 - PATCH /students/34322
 - Example: 34322 is student id
 - DELETE /students?billingStatus=overdue

URL: Path Parameters

- Resource path often has subpaths
 - Represents more specific resource
- GET /students
 - Read resource: all students
- GET /students/34322
 - Read resource: student with id 34322
- GET /students/Naresh/Rajkumar
 - Read resource: student Naresh Rajkumar

In all cases, Path represents the resource

Second Rule of REST

HTTP method is the interaction with the resource

- The resource 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

Details of changes passed in data, 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; Create new one
- PUT (Replace)
 - Replace existing record
 - Save nothing from existing record
- 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/Kumar Overwrite
 - Send: (data to replace with)
 - Get: (usually updated record)
- PATCH /students/Naresh/Kumar Partial Update
 - Send: (fields with changed values)
 - Get: (usually updated record)

Third Rule of REST

HTTP Status code is interaction result

- There are many Status codes!
 - With meaningful names
 - A few are misleading
 - Confirm the meaning/use (MDN)
- Add details in body
 - Status alone often not enough
 - What format will you use for details?

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 (**5**xx): 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)
 - Common point of confusion (more soon)
- 500 (Internal Server Error) Server had issue
 - Not user's fault
 - Not expected!

Common issues with some REST status codes

- 404 (Not Found)
 - API Path wrong?
 - OR API Path right, but that data doesn't exist?
 - vs a 200 w/an empty data ({} or [])?
 - Service calls should not return html pages
- 204 (No Content)
 - Could be returned by a POST/PUT/PATCH
 - Saves on bytes sent
 - Makes parsing service results harder

REST Response

- REST says: HTTP Status code
 - REST doesn't say much else about response
- Common responses (Can vary!)
 - If server created a UUID/ID for new resource
 - Provide new record or URL in response
 - 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 = { Jorts: { name: 'Jorts', age: 3 } };
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]); // default 200 OK status
        return;
    }
    res.status(404).json({ error: `Unknown cat: ${name}`});
});
```

- :name syntax (express) sets req.params.name
 - example: GET /cats/Jorts
- res.json() does JSON.stringify()
 - AND sets 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; // Poor Security! Unsanitized!
    res.sendStatus(204); // "No content"
  }
});
```

express.json() middleware populates req.body

• Parses request content-type of application/json

No request content-type means no req.body value

Creating a basic REST service in express

- Route Path that matches Rule 1
- Use a method that matches Rule 2
- Use correct status codes for Rule 3
- Check for any auth requirements!
 - Same req.cookie/sid checks
- Parse incoming body data
 - express.json() for JSON
- Set res. status if not 200
- Send JSON data in response
 - res.json()
- No HTML, No Redirects

REST in Express: REST Rules

- Route Path that matches Rule 1
 - URL Path is resource
 - app.XXX('/cats/:name', (req, res) => {
- Use a method that matches Rule 2
 - Method = Interaction with Resource
 - app.get('/cats/:name', (req, res) => {
 - app.post('/cats/:name', (req, res) => {
- Use correct status codes for Rule 3
 - Status Code = Interaction Result
 - res.status(409)
 - res.sendStatus(204)

REST in Express: Auth

- Check for any auth requirements!
 - On ALL requests that expect auth'ed user
 - Same req.cookie/sid checks
 - But no redirect/login form if bad sid!
 - Send correct status if bad sid
 - 401 (Auth Missing)
 - If no sid/bad sid
 - 403 (Forbidden)
 - If valid sid but not allowed
 - We also do this for user "dog"

REST in Express: Parsing Request Data

- Resource identifiers from URL path in req.params
- Parse incoming body data
 - express.json() for JSON
 - Sanitize incoming data! (Security!)
- Set appropriate status if data has problems
 - 400 (Bad Request)
 - General "user sent bad data" response
 - Provide details in response body
 - 409 (Conflict)
 - User request conflicts with existing data

REST in Express: Sending Response

- If status is not 200 (Success)
 - Set res.status BEFORE .send()/.json()
 - Can't change/set status after response sent!
- Send JSON data in response
 - res.json()
 - Sets content-type header for response
 - Stringifies and sends JSON Body
 - I recommend JSON for both errors/success
 - Makes parsing in client easier
- No HTML, No Redirects
 - Service response is not a page response

Writing a REST Service

- Service is entirely state/data changes!
 - No presentation! No HTML View!
 - Server MVC pattern basics still valid
 - "Model" updates just as important
 - "View" now subset of state to return
 - Often not exact actual state
 - Only the data the client needs
 - May not be separate file
 - Still a separate concept
- Server state and Client state NOT the same
 - Often similar, not the same