# REST APIs Do’s and Don’t’s

## Audience

- Developers already building APIs
- Developers looking to start building APIs

## Agenda

- What is REST?
- How does it compare to SOAP? gRPC? GraphQL?
- Do’s and Dont’s
- Questions any time

## Goals

- Expose you to what REST is
- Expose you to ideas to avoid with REST

## What is REST?

- Representational State Transfer
- Stateless communication – no sessions
- Resource based design (ie /users, /users/123)
- Leverage HTTP standards to communicate requests and responses
- JSON

## How does REST compare?

| | REST | SOAP | gRPC | GraphQL |
|---|---|---|---|---|
| Protocol | HTTP | XML | HTTP/2 | HTTP (single endpoint) |
| Format | JSON | XML | Protobuf (binary) | JSON (w/ query lang) |
| Ease of use | Simple, widely used | Strict Standards | Proto file and tools | Schema + queries |
| Performance | Good, verbose | Slower due to XML size | Highest performance | Good, what you need |
| State | Stateless | Stateful or Stateless | Stateless | Stateless |
| Operations | CRUD via HTTP methods | Predefined operations | RPC styl | Client defined |
| Flexibility | Moderate | Low | Moderate-High | Highest |
| Error Handling | Status Codes | Defined errors | Status codes in metadata | Custom error structure in response |
| Usecases | Web APIs, CRUD apps | Legacy enterprise | High performance | Lots of clients |
| Adoption | Very high | Declining | Growing slightly | Growing faster |

## When would you use REST?

- Want an industry standard
- Clients need less flexibility
- Safe choice
- Caching
- JSON is more efficient than XML (SOAP)

## When would you use SOAP?

- Only if you absolutely have to
- 3rd party API that only supports SOAP

## When would you use gRPC?

- Absolute need for high performance (HTTP/2, Binary instead of JSON)
- Real time communication
- Need streaming support (ie for large file transfers, continuous data streams)

## When would you use GraphQL?

- GraphQL allows you to choose which fields you want to return
- Perfect for lots of clients with unknown data needs
- Less pressure on your database
- This is called “sparse fieldsets”

## Core Design Principles of REST API

- URLs are your Resources
- Status Codes communicates result of operation
- HTTP Methods/Verbs communicate Request’s intent

## HTTP Methods/Verbs

- GET == SELECT
- POST == INSERT
- PATCH == UPDATE some columns
- PUT == UPDATE all columns
- DELETE == DELETE

## URLs determine resources

- /users = Manage Users
- /employees = Manage Employees
- /orders = Manage Orders

## Status Codes communicate result status

- 2xx = Success
- 4xx = The caller of the API screwed up
- 5xx = The API screwed up

## Common 2xx Status Codes

- 200 OK = everything was fine
- 201 Created = usually for POSTs to say it worked
- 202 Accepted = usually for eventually consistency
- 204 No Content = usually for DELETEs to say it worked

## Common 4xx Status Codes

- 400 Bad Request = You sent me bad data (ie missing required fields)
- 401 Unauthorized = You forgot to sent me a JWT or API Key
- 403 Forbidden = You sent me valid credentials, but you can’t call this API
- 404 Not Found = You called an invalid endpoint
- 418 I’m a teapot = Very critical if you’re Starbucks

## Common 5xx Status Codes

- 500 Internal Server Error = Something went wrong, usually unhandled exception
  - **DO NOT EXPOSE STACK TRACES OR ANYTHING TO CLIENTS, THAT'S WHAT LOGS ARE FOR**
- 503 Service Unavailable = IIS or App Service having an issue

## Example

- Have a User Management page to manage users
- GET to /users
  - Returns back all users (possibly paginated)
  - 200 OK means everything was good and data is in the body
  - Also possible 401, 403
- GET to /users/abc-123
  - Returns back just user with ID of abc-123
  - 200 OK means everything was good and data is in the body
  - Also possible 401, 403
  - 404 means that ID was not found

## Example

- POST to /users with body { firstName: “Bob”… }
  - 201 Created with body { id: “abc-123” }
  - Also possible 401, 403
- PATCH to /users/abc-123 with body { middleName: “Edward” }
  - 200 OK means everything was good and data is valid in the body, just updating middle name to Edward
  - Also possible 401, 403
  - 404 means that ID was not found
- PUT to /users/abc-123 with body { firstName: “Bob”, lastName: “Smith”…}
  - 200 OK means it worked updating all fields
  - Also possible 401, 403
  - 404 means that ID was not found

## Slightly more complex example

- Announcements
- I need Admins to manage announcements, but I need non-Admins to view targeted announcements (ie for different types of customers)

## Example

- GET to /announcements
  - Returns back all announcements (possibly paginated) for Admins only
  - 200 OK means everything was good and data is in the body
  - 401 unauthorized, no JWT
  - 403 unauthorized, user is not an admin
- GET to /announcements/mine
  - Returns back single announcement for current user (note: Admins can be users)
  - 200 OK means everything was good and data is in the body
  - 401 unauthorized, no JWT

## Example

- POST to /announcements
  - 201 Created with body { id: “abc-123” }
  - Also possible 401, 403 (not an admin)
  - This might not only create a new announcement, but expire/remove an old one.

## Lessons

- Avoid APIs that are just an endpoint per table
- Might be tempting to do /announcements to return all announcements then filter client-side
- But that data will get leaked to clients
- Enforce business rules server side
- Map APIs to business processes
- Beware of “I need these 3 API calls to succeed”
- What if API call 1+2 succeeds but the 3rd fails?

## Additional Gotchas

- Too many systems talking directly to the database? Use an API so one system talks to the database
- 1-3 systems is probably okay
- More than that is probably a 🚩
- But an API adds overhead, network, dev time, logging/tracing, etc

## What kind of API are you building?

- Knowing what kind of API you’re building brings clarity to your design
- Process API
- Experience/BFF API

## Process API

- Typically consumed by other APIs
- Meant to be reusable
- Goal is to abstract a Business Process
- Might talk to multiple systems so every consumer doesn’t have to
- Could be inventory data for a warehouse, pricing information from a catalog, etc.

# Experience/BFF API

- Backend for Frontend not Best Friends Forever
- API specific to 1 particular client/frontend
- …so kinda like the API and frontend are BFFs
- Not meant to be reusable
- Tailors the backend interface to exact needs of that frontend
- Optimized to reduce frontend complexity and exact needs of that client

## Process vs BFF APIs

| | Process | BFF |
|---|---|---|
| Purpose | Data aggregation, orchestration | Client-specific data optimization |
| Audience | Other systems/APIs | Specific frontend (ie web/mobile) |
| Reusability | High | Low |
| Coupling | Loosely coupled | Tightly coupled to the frontend |
| Complexity | Reduces backend complexity | Reduces frontend complexity |

## Versioning

- Sometimes you need to introduce breaking changes
- Versions allow you to do that while remaining backwards compatible
- ie /v1/announcements, /v2/announcements
- Can also do query string, header, or content type
- Start with a Version if you have a Process API, if it’s a BFF API, likely unnecessary (for web apps)
- Don’t re-use models between versions (default values)
- https://www.nuget.org/packages/Asp.Versioning.Mvc/

## API Gateways

- All requests go through API Gateway first
- Allows for logging
- Allows for reusable logic everyone needs (ie validate a JWT)
- Allows for custom logic

## Takeaways

- Understand what REST is
- When to use it
- Common Do’s and Don’t’s
- Things to consider for APIs (design, versioning, API GW, etC)

What situations do you guys have?

Questions?