W06-1: Microservices, APIs, and REST

Dr. Alex Adkins

CPSC 3720: Software Engineering

CUSports Project: Sprint 0

Sprint 0 (25 points):

- Team Kickoff and CUSports epics using the Trello Board instructions
- Use the Kickoff and Epic boards in your Trello Workspace
- 2-3 minute Sprint Review per team Thursday 2/13 to present:
  - Team Kickoff Board with team name and logo
  - Epic and Services board

- **5 points** for thoughtful kickoff board/name/theme/logo
- **10 points** for thoughtful Epics and Services
- **5 points** for the Sprint Review
  - if you are not present you lose the points
- Team Survey due at end of day Friday 2/14
  - if you don't do the survey, it will impact your grade!

Sprint 0 Review

- All team members MUST be present for Team Kickoff review
  - Absent students will get a 0 for the review

- 3-minute review max per team to present:
  - Team name, theme, and logo
  - Team Kickoff Board
  - Epics/Estimates for a designated role
    - I’ll let you know which role to share these for
  - Where did you draw your prioritization line?

- Be enthusiastic!

Today’s Objectives

By the end of class today, you should understand:

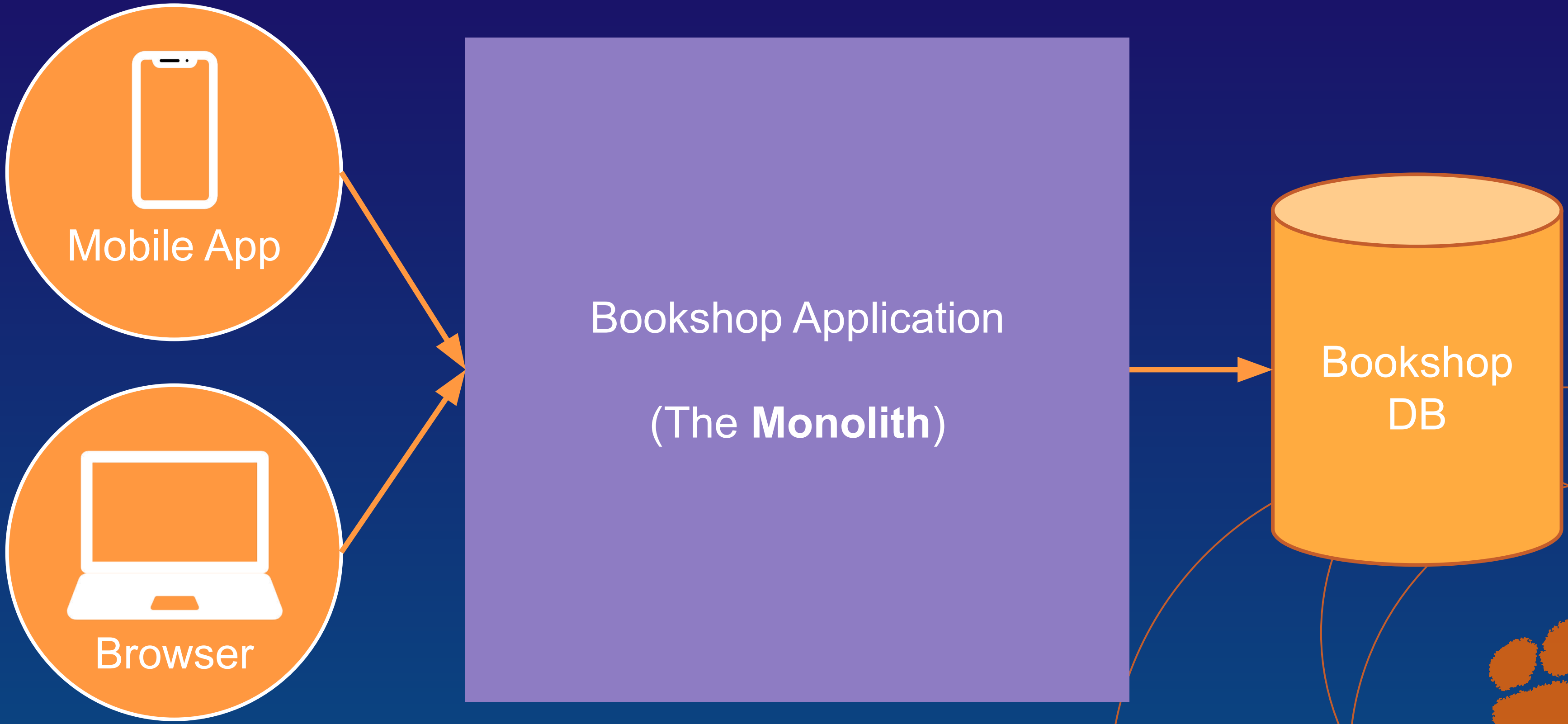
- **Microservices**
  - Microservice-based architectures at a high-level
  - How this architecture is relevant to our class project
- **APIs**
  - Response & Request Model
  - REST endpoints
  - HTTP methods

Microservices

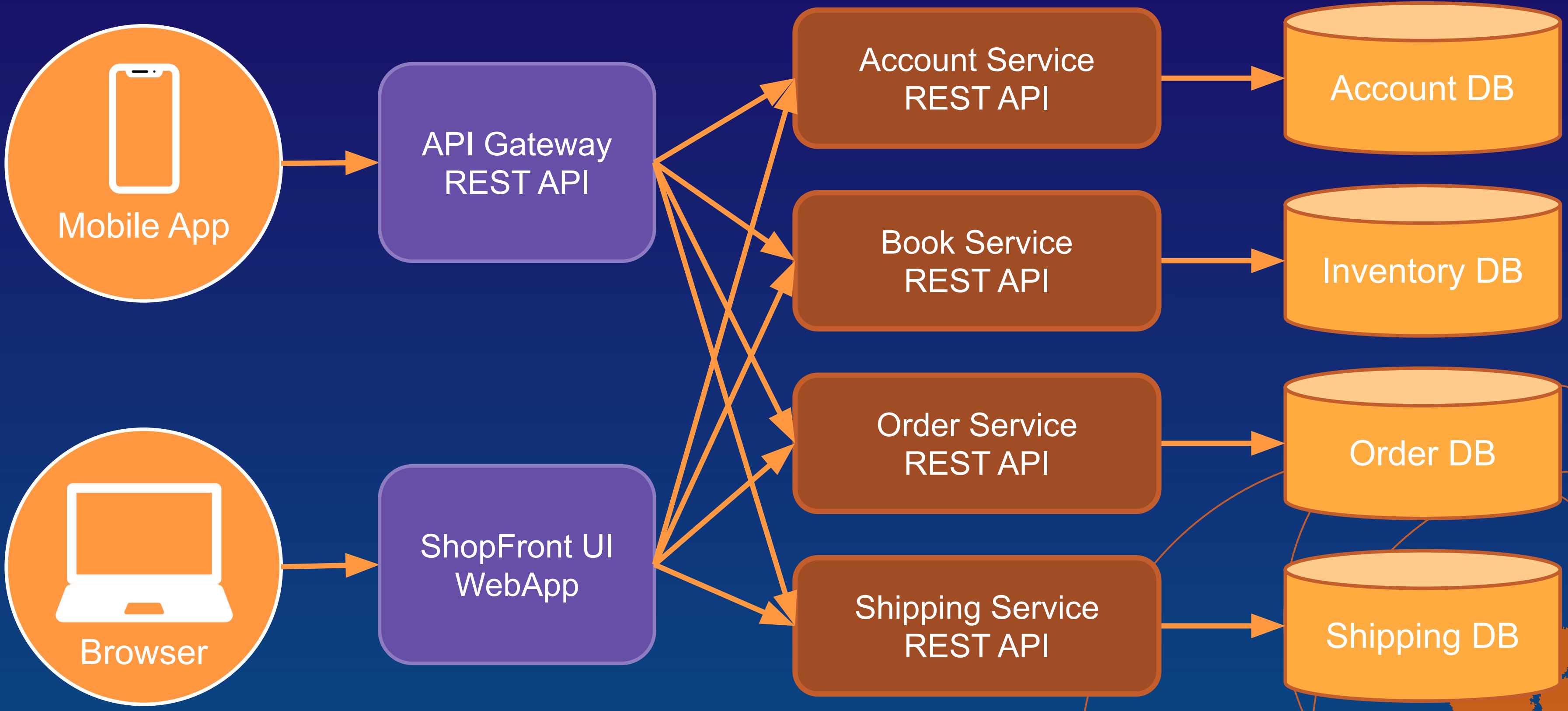
Companies that moved to Microservices:

amazon
Coca-Cola
UBER
airbnb
NETFLIX
ebay
PayPal
in

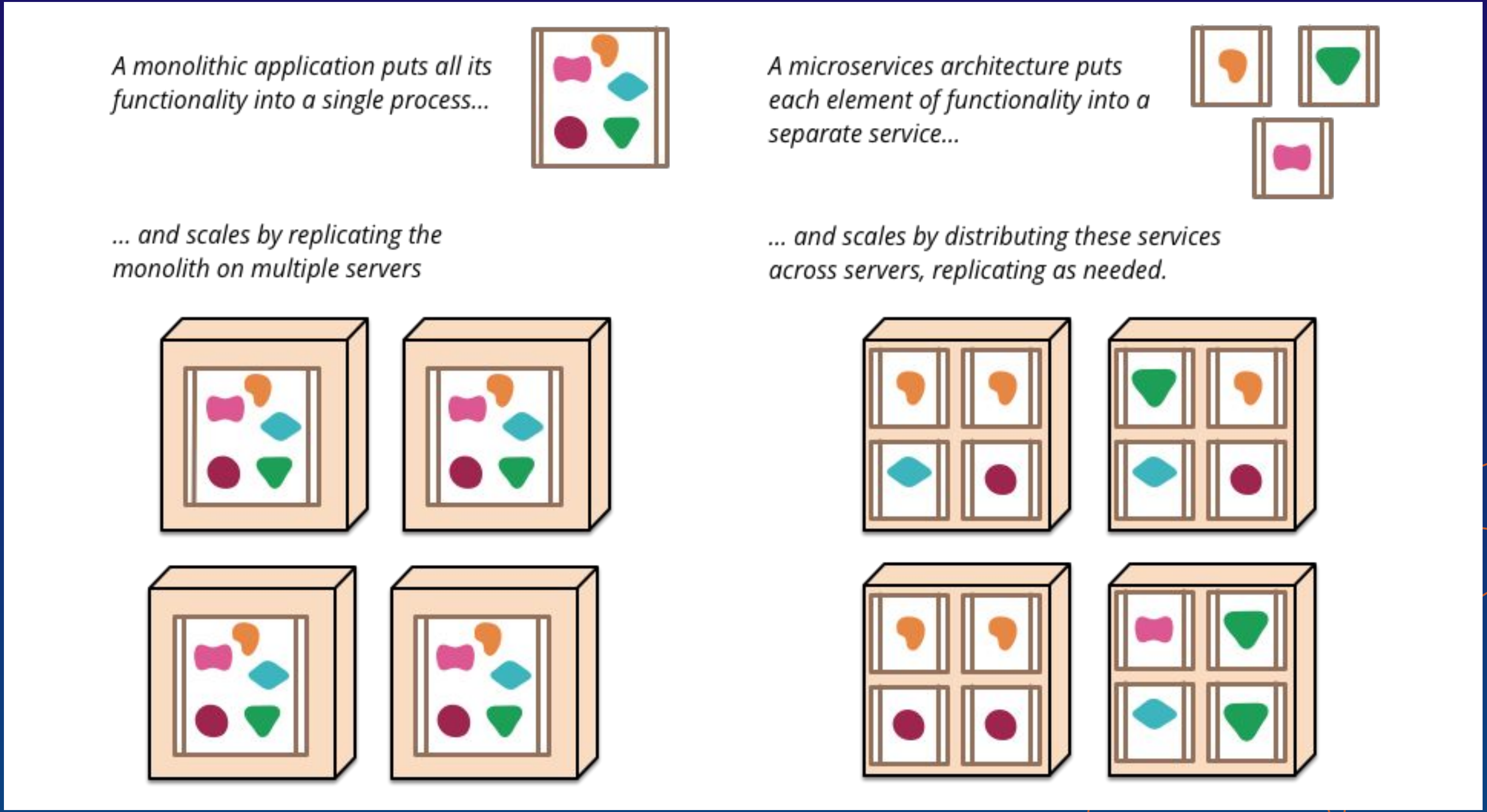
Before Microservices: Online Bookshop



Microservices Example: Online Bookshop



Applications: From Monoliths to Microservices



A monolithic application puts all its functionality into a single process...

A microservices architecture puts each element of functionality into a separate service...

... and scales by replicating the monolith on multiple servers

... and scales by distributing these services across servers, replicating as needed.

https://martinfowler.com/articles/microservices.html

Why Microservices?

- Expectations have changed regarding the delivery of software:
  - Rapid-change and rapid delivery
  - High scalability and reliability
  - Cloud-based
- **Microservice**-based architectures enable the incredible scale and agility needed in the software solutions today

Why Microservices?

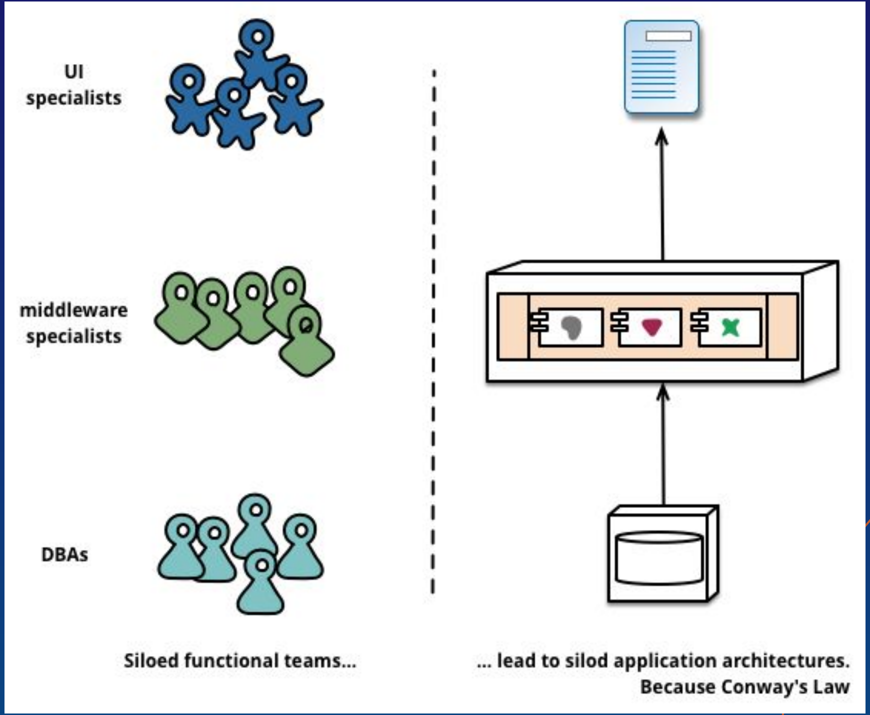
- Greater agility
- Continuous integration and deployment
- Improved scalability
- Faster time-to-market
- Higher developer productivity
- Easier debugging and maintenance

Microservice Characteristics

- Decentralized Governance
  - Each service independently chooses tech and standards
- Decentralized Data Management
  - Services manage their own databases separately
- Evolutionary Design
  - Architecture adapts to changing requirements over time
- Infrastructure Automation
  - Deployment and scaling handled automatically
- Design for Failure
  - Services expect and recover from failures gracefully

Conway’s Law in Action

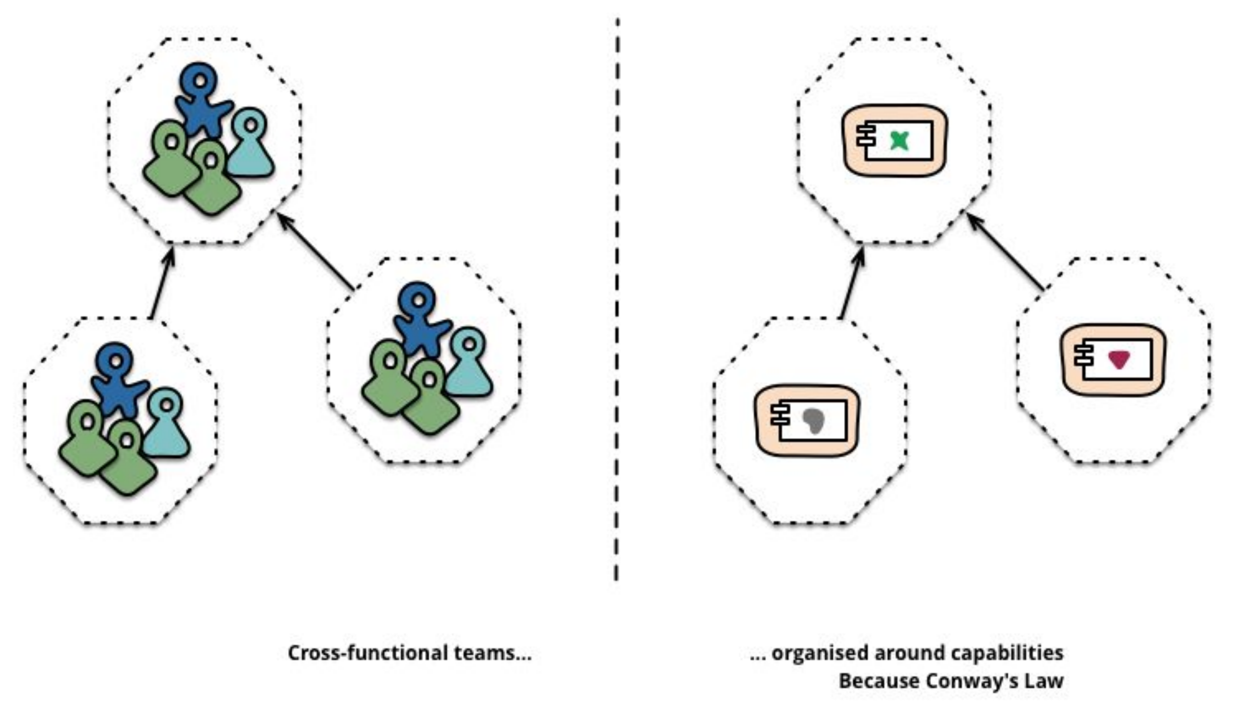
“Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.”


UI specialists
middleware specialists
DBAs
Siloed functional teams...
... lead to silod application architectures.
Because Conway's Law

https://martinfowler.com/articles/microservices.html

Microservices should Align with Business Capabilities

Teams are organized around *services*



https://martinfowler.com/articles/microservices.html

Microservice Attributes

- Microservices aim to be as **decoupled** and as **cohesive** as possible - they own their own domain logic
- Microservices use HTTP request-response with resource **API's** and lightweight messaging; this requires **service contracts**.
- Each microservice typically has its own database

Coupling & Cohesion

Microservices aim to be as **decoupled** and as **cohesive** as possible - they own their own domain logic.

High Cohesion

Low Coupling

Cohesion

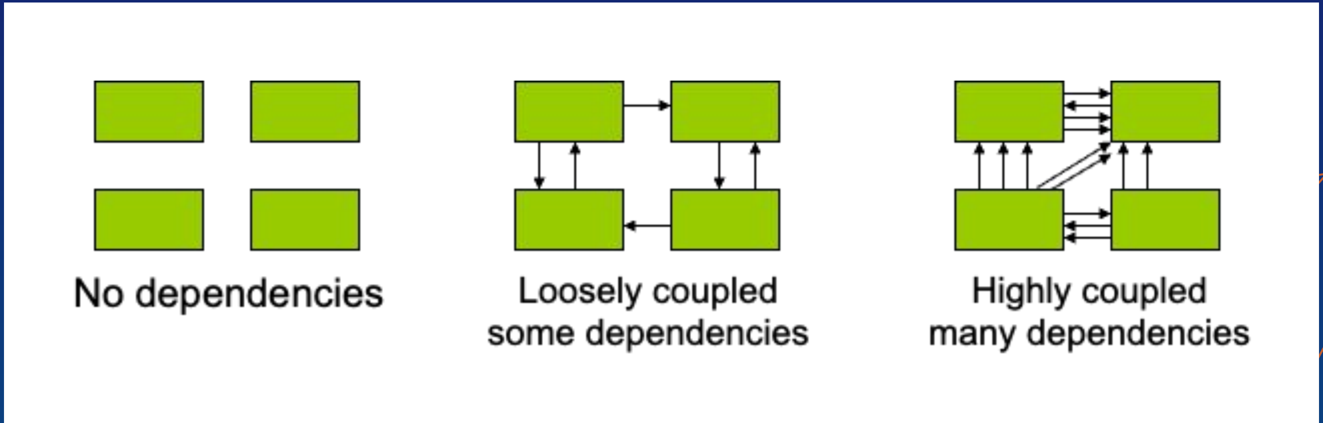
Definition

- The degree to which all elements of a component are directed towards a single task.
- The degree to which all elements directed towards a task are contained in a single component.
- The degree to which all responsibilities of a single class are related.

High Cohesion: All elements of a component are directed toward and essential for performing the same task.

Coupling

The degree of dependence across components such as the amount of interactions among components



Discussion

At your tables…

What is the effect of **cohesion** and **coupling** on maintenance?

- **Cohesion**: All elements of a component are directed toward and essential for performing the same task
- **Coupling**: The degree of dependence across components such as the amount of interactions among components

Consequences of Coupling

- High coupling
  - Components are difficult to understand in isolation
  - Changes in component ripple to others
  - Components are difficult to reuse
  - Need to include all coupled components
  - Difficult to understand
- Low coupling
  - May incur performance cost
  - Generally faster to build systems with low coupling

Example…

Clemson Online Bank: Epics

- As a bank user, I want to login to my account.
- As a protentional bank customer, I want to signup for an account.
- As a bank user, I want to edit my account information.
- As a bank user, I want to transfer money to another Clemson Bank account
- As a bank user I want to transfer money within my own account.
- As a bank user, I want to transfer money to a non-Clemson account.
- As a bank user, I want to be notified if I have insufficient funds in my account.
- As a user, I want to deposit a check online
- As a bank user, I want to set up travel notifications
- As a bank user, I want to receive notifications of suspicious transaction activity.
- As a bank user, I want to see dashboards of my spending.
- As a bank user, I want to replace a credit card that I lost.
- As a bank user, I want to lock or unlock credit cards.
- As a bank user, I want to see transaction history.

Discussion

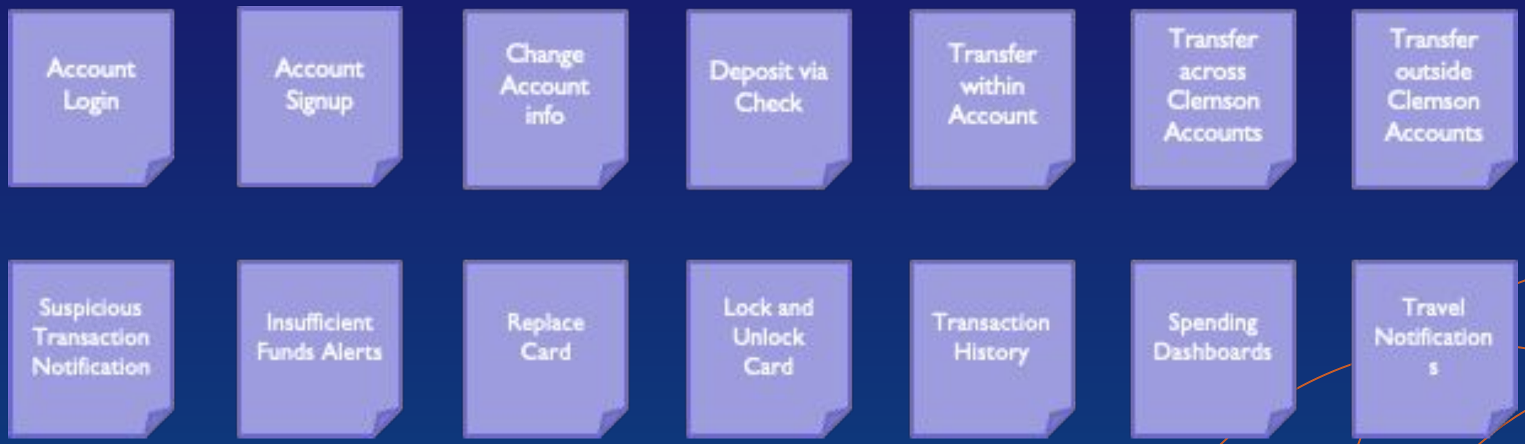
At your tables…
(10 minutes)

How would you break up the Clemson Bank into Services?

Work with your teams and brainstorm a list of “Services” that would make up the component/microservices of the Bank to support these Epics. Have a list to share in 10 minutes.

Example…

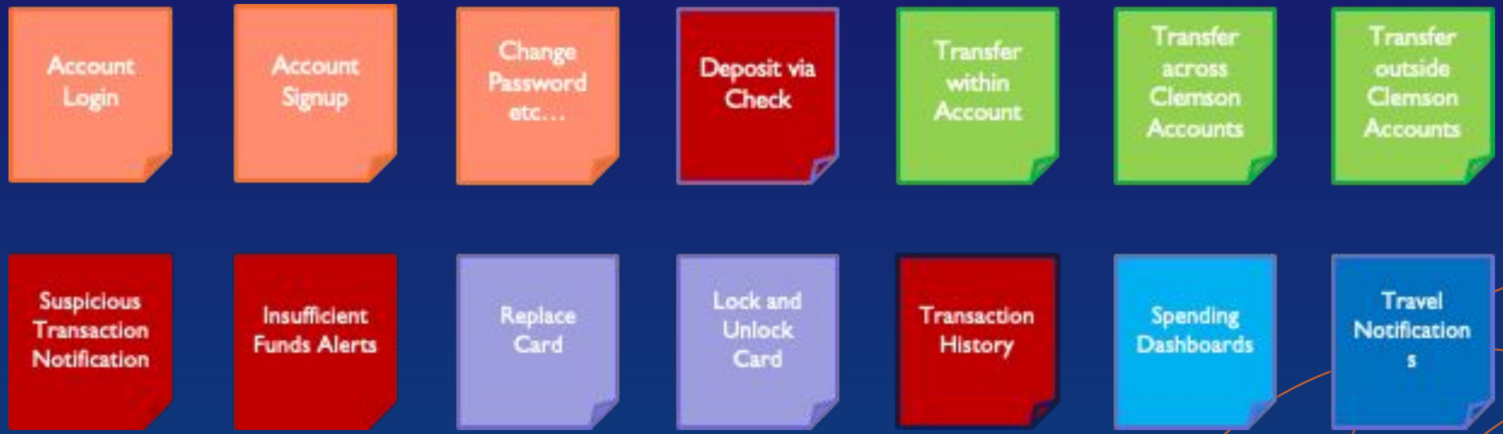
Clemson Online Bank: Epics



- Account Login
- Account Signup
- Change Account info
- Deposit via Check
- Transfer within Account
- Transfer across Clemson Accounts
- Transfer outside Clemson Accounts
- Suspicious Transaction Notification
- Insufficient Funds Alerts
- Replace Card
- Lock and Unlock Card
- Transaction History
- Spending Dashboards
- Travel Notifications

Group Similar Requirements…

Clemson Online Bank: Epics



Account Login

Account Signup

Change Password etc…

Deposit via Check

Transfer within Account

Transfer across Clemson Accounts

Transfer outside Clemson Accounts

Suspicious Transaction Notification

Insufficient Funds Alerts

Replace Card

Lock and Unlock Card

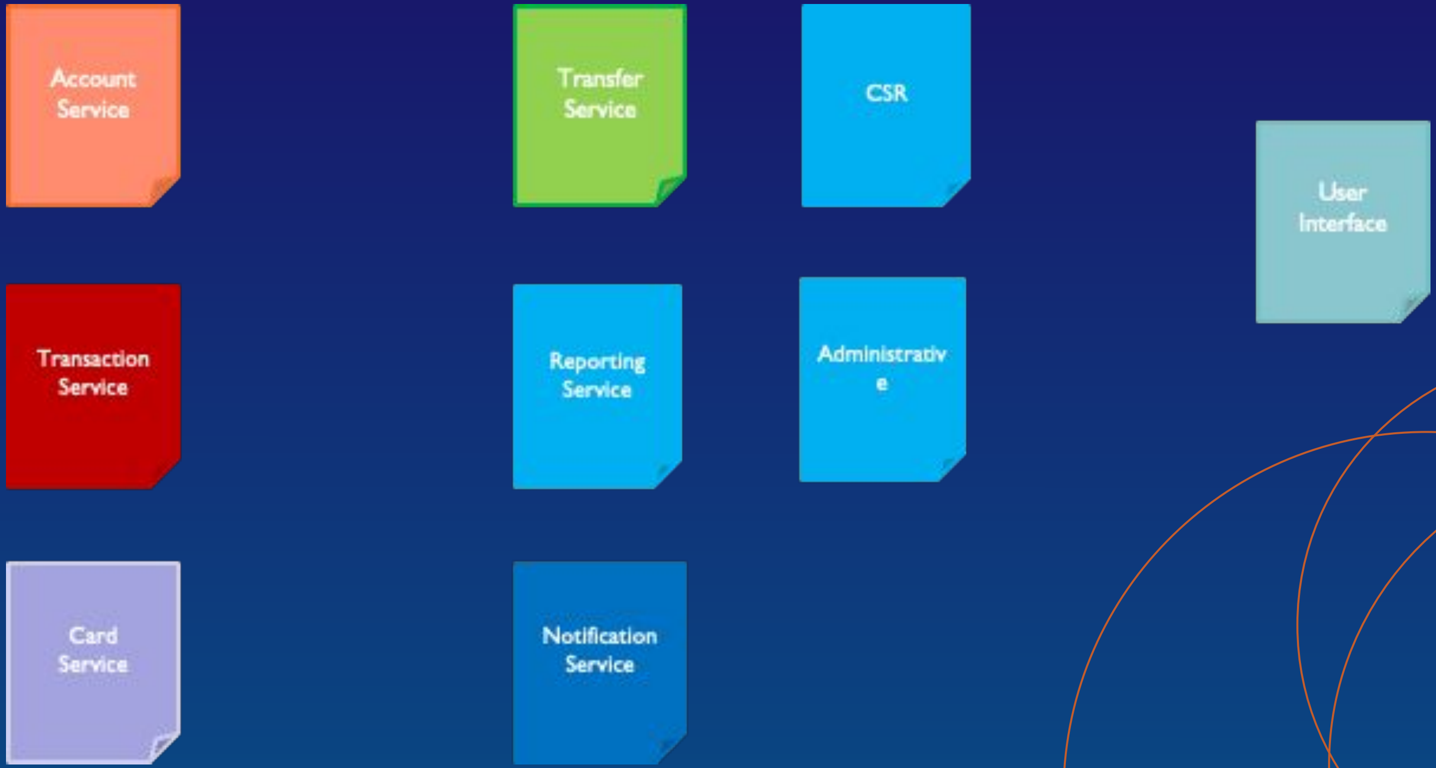
Transaction History

Spending Dashboards

Travel Notification s

Services!

Clemson Online Bank: Services



Microservice Attributes

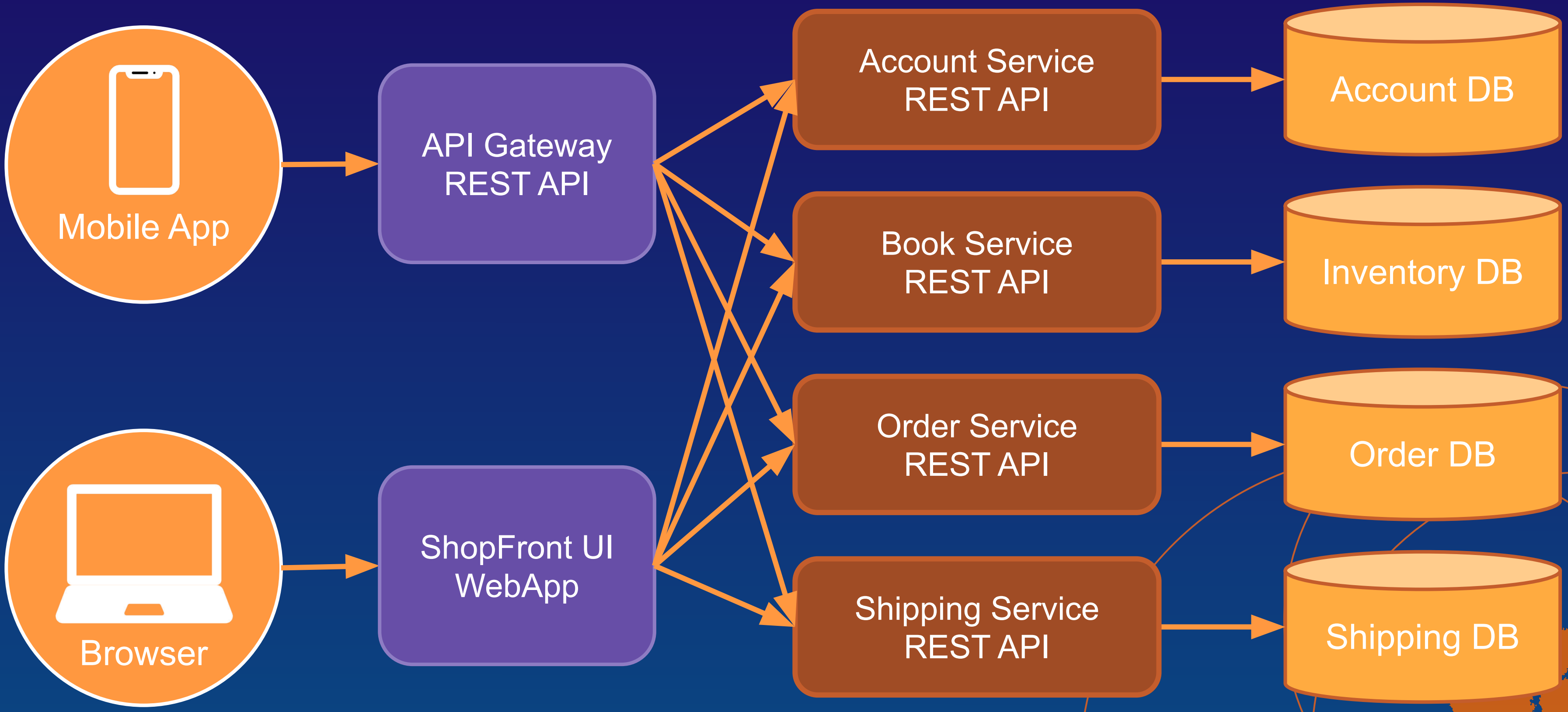
- Microservices aim to be as **decoupled** and as **cohesive** as possible - they own their own domain logic

- **Microservices use HTTP request-response with resource API's and lightweight messaging; this requires service contracts.**

- Each microservice typically has its own database

REST APIs

Microservices Example: Online Bookshop



Microservice Attributes - Review

- Microservices: aim to be as **decoupled** and as **cohesive** as possible
  - Decoupled: each service should operate independently
  - Cohesive: each service focuses on a specific domain or business logic

- Microservice Communication: uses the **HTTP request-response** model
  - Services interact through resource APIs
  - The use lightweight messaging for asynchronous communication

- Messaging Infrastructure: typically **simple**, only acts as a message router
  - **The "Smarts" are in the services**
    - Services are the endpoints process simple messages and respond accordingly

- Accessing Microservices: through **APIs**

APIs: Application Programming Interfaces

- API: Application Programming Interface
  - A set of communication rules for software
  - Defines structure of *requests* and *responses*
- Request-Response Model
  - A client makes a *request* to a service
  - The service processes the request and send back a *response*
- API types:
  - **REST**
    - Uses HTTP
    - Most common
  - SOAP
    - Uses XML
  - GraphQL

A RESTful World of Services

- REST: Representational State Transfer
  - completely changed software engineering after 2000.
- Defined by Roy Fielding
  - Father of the HTTP specification, in his dissertation entitled "Architectural Styles and the Design of Network-based Software Architectures"
  - HTTP: HyperText Transfer Protocol
- REST today:
  - the "be all-end all" in service app development
  - Most software projects and applications expose REST APIs for the creation of services based on this software
  - Examples: X, YouTube, Facebook
  - Hundreds of companies generate business thanks to REST
- So, what is REST?

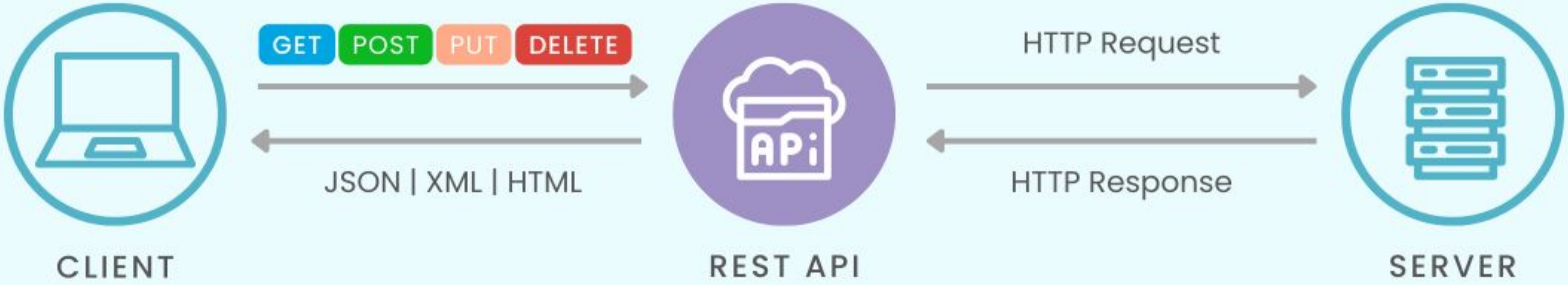
What are REST APIs?

https://www.youtube.com/watch?v=SLwpqD8n3d0

What is REST?

- REST: Defines how an API should look and function
  - Set of rules developers follow when creating APIs
- Resources & URLs:
  - Resource: A piece of data that can be accessed
    - e.g. a user, an order, a product
  - URL: Unique address to a specific resource
    - URL: Uniform Resource Locator
    - Each URL call is called a request
- Request and Response model:
  - Request: Client sends a URL request to access or modify a resource
  - Response: The server sends back the requested data or a status message

REST API MODEL



REST Requests

Components of REST Requests

- The REST request is a URL made of four components:

    1. The endpoint
    2. The method
    3. The headers
    4. The data (or body)

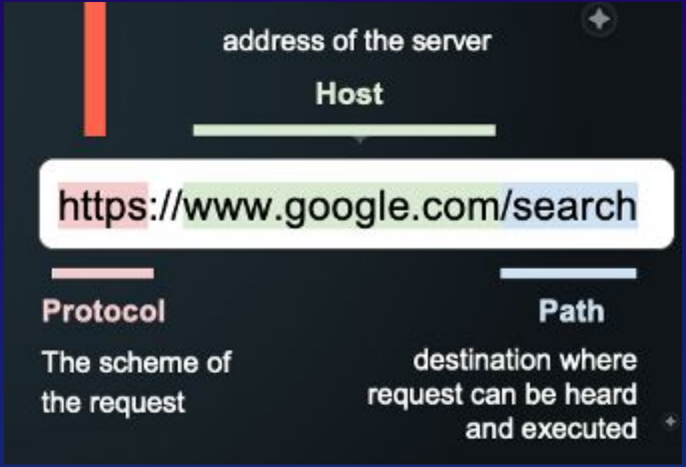
Components of REST Requests

- The REST request is a URL made of four components:

  1. **The endpoint**
  2. **The method**
  3. The headers
  4. The data (or body)

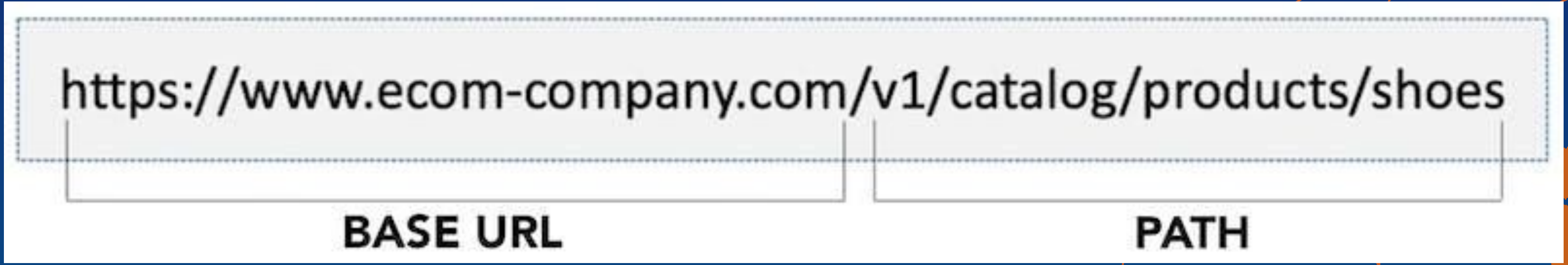
REST API Endpoints

- API Endpoint: the specific location/access point to the API
    - Represented as URLs


address of the server
Host
https://www.google.com/search
Protocol
The scheme of the request
Path
destination where request can be heard and executed

Endpoint Components:

- Base URL / Root-endpoint: points to the host
- Path: points to the API



Understanding REST Endpoints

Base URL/Root Endpoint: The starting point of the API Request

- Examples:
  - GitHub API: https://api.github.com
  - Twitter API: https://api.twitter.com

Path: Determines the specific resource being requested

- Example:
  - URL: https://www.smashingmagazine.com/tag/javascript/
  - Root-endpoint: https://www.smashingmagazine.com/
  - Path: /tag/javascript

REST Endpoint Path Specificity

Query Parameters:

- Start with `?`
- Represent key-value pairs joined with `&`
- Example:
  - `.../search?q=cat`
  - `.../search?query=nature&orientation=landscape`

Path Parameters:

- Represent dynamic elements of a path
- Often include an ID value
- Example:
  - `.../photos/j1hc6C3gO0Q`
  - `.../photos/_UHlujD2Jwo`

REST Endpoint Paths

Endpoint Paths:

- Find an endpoint’s available paths in the API Documentation

Github API Example:

- To get a list of repositories by a user in Github’s API, check Github Docs for the correct path:
  - `/users/:username/repos`
- `:` indicates a variable in the path you replace with real info
- `:username` is a variable you replace with a real user:
  - `https://api.github.com/users/alexadkins/repos`

REST Endpoint Query Parameters

Query Parameters: allow you to modify your request with key-value pairs.

- Though not part of the REST architecture, many APIs use them.

Format:

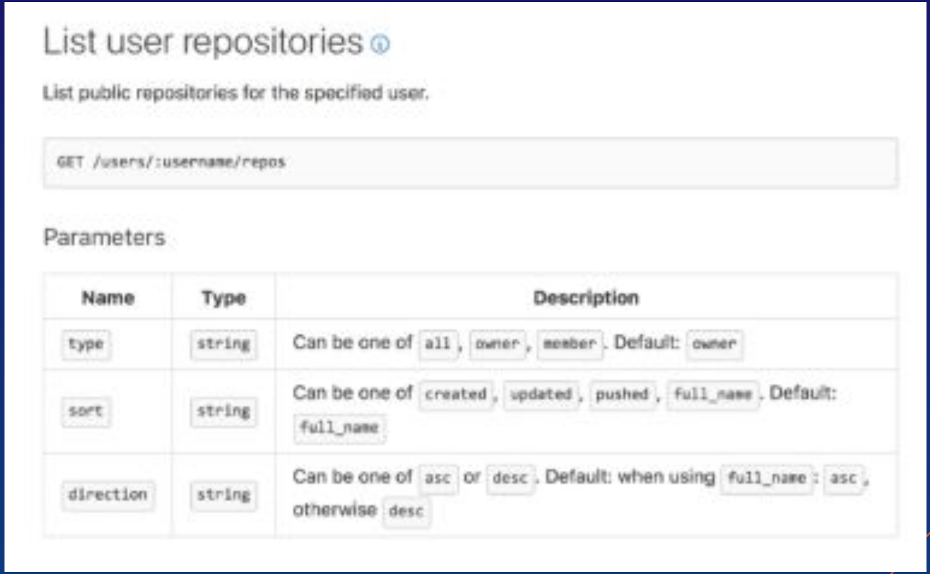
- Begins with a ? followed by key-value pairs
- Each pair is separated by an &

Example:

- `?query1=value1&query2=value2`

REST Endpoints Query Parameters

Check API documentation for Query Parameter options:



List user repositories

List public repositories for the specified user.

```
GET /users/:username/repos
```

Parameters

Name	Type	Description
`type`	`string`	Can be one of `all`, `owner`, `member`. Default: `owner`
`sort`	`string`	Can be one of `created`, `updated`, `pushed`, `full_name`. Default: `full_name`
`direction`	`string`	Can be one of `asc` or `desc`. Default: when using `full_name`: `asc`, otherwise `desc`

Example request:

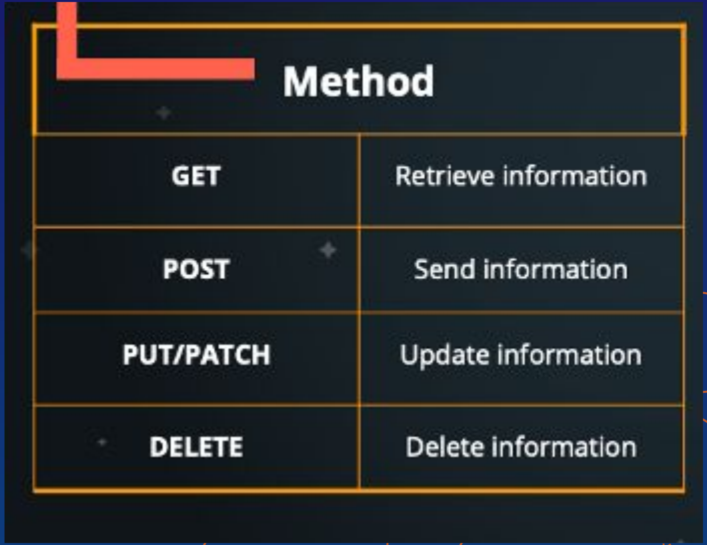
`https://api.github.com/users/alexadkins/repos?sort=pushed`

HTTP Methods

- HTTP Methods: actions you can take when interacting with an API
  - Technically 8 types; only 4 commonly used

HTTP Methods:

- **GET**: retrieve information
- **POST**: send information
- **PUT/PATCH**: update information
- **DELETE**: delete information



Method	
GET	Retrieve information
POST	Send information
PUT/PATCH	Update information
DELETE	Delete information

HTTP METHOD	DESCRIPTION
GET	This request is used to get a resource from a server. If you perform a `GET` request, the server looks for the data you requested and sends it back to you. In other words, a `GET` request performs a `READ` operation. This is the default request method.
POST	This request is used to create a new resource on a server. If you perform a `POST` request, the server most likely creates a new entry in the database and tells you whether the creation is successful. In other words, a `POST` request performs a `CREATE` operation.
PUT & PATCH	These two requests are used to update a resource on a server. If you perform a `PUT` or `PATCH` request, the server updates an entry in the database and tells you whether the update is successful. In other words, a `PUT` or `PATCH` request performs an `UPDATE` operation.
DELETE	This request is used to delete a resource from a server. If you perform a `DELETE` request, the server deletes an entry in the database and tells you whether the deletion is successful. In other words, a `DELETE` request performs a `DELETE` operation.

Components of REST Requests

- The REST request is a URL made of four components:

    1. The endpoint
    2. The method
    3. **The headers**
    4. The data (or body)

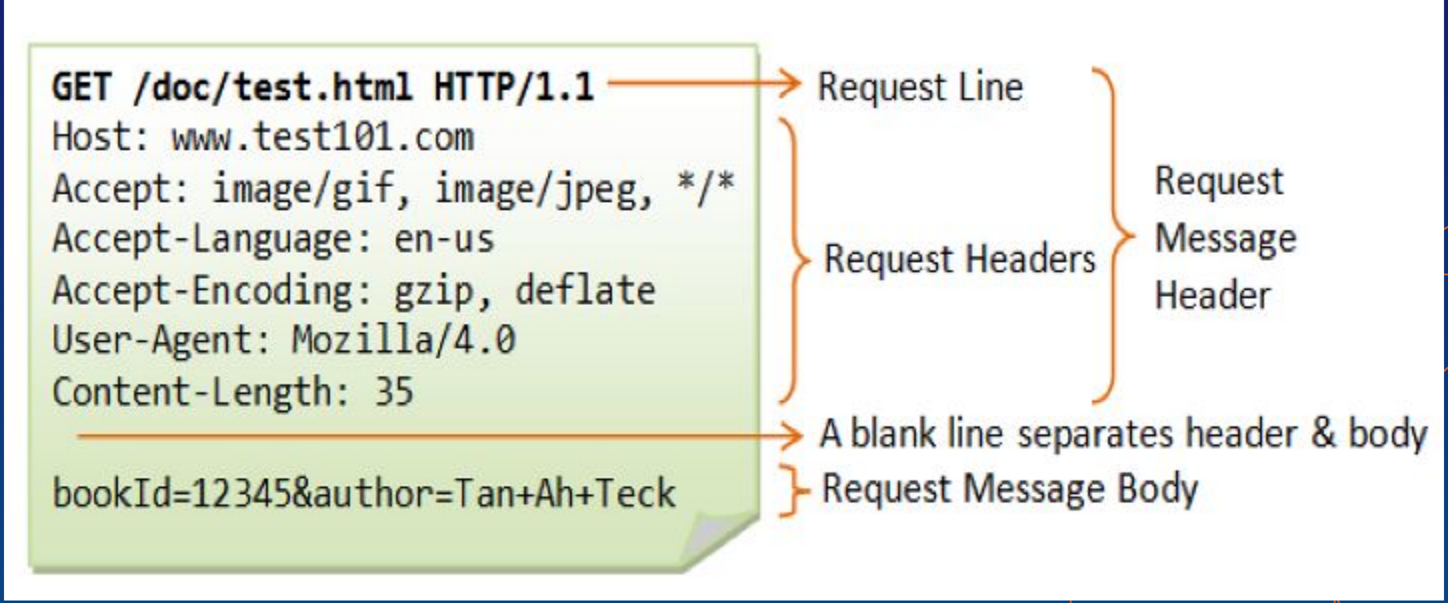
REST Headers

- Headers: Key-value pairs for non-payload (body) information
  - Provides metadata

- Common standard headers:
  - Authorization
  - Cookie
  - Content-Type
  - Accept
  - User-Agent

REST Headers

- Used to provide information to both client and server
- Find a list of valid headers on MDN’s HTTP Headers Reference
- HTTP Headers are property-value pairs separated by a colon



```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

Components of REST Requests

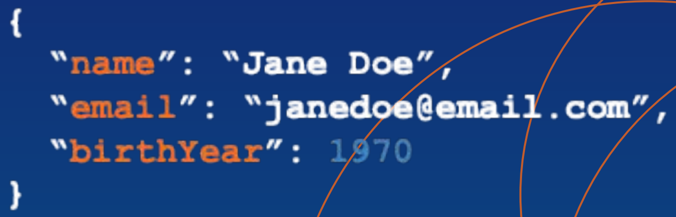
- The REST request is a URL made of four components:

    1. The endpoint
    2. The method
    3. The headers
    4. **The data (or body)**

REST Data Payload (aka “Body”)

- Data Payload: contains information you want to be sent to the server
  - Used with POST, PUT/PATCH, or DELETE
- Data types:
  - form data
  - JSON
  - text
  - HTML
  - XML
  - files
  - GraphQL
  - more!

JSON:



```
{
    “name”: “Jane Doe”,
    “email”: “janedoe@email.com”,
    “birthYear”: 1970
}
```

Amazon Request Body Example

```
POST https://products.amazon.com/api/2017/add
{
  "product_name": "Smartphone",
  "price": 499.99,
  "description": "A smartphone with advanced features.",
  "category": "Electronics",
  "availability": true
}
```

Components of REST Requests

- The REST request is a URL made of four components:

    1. The endpoint
    2. The method
    3. The headers
    4. The data (or body)

What about the RESPONSE from the API?

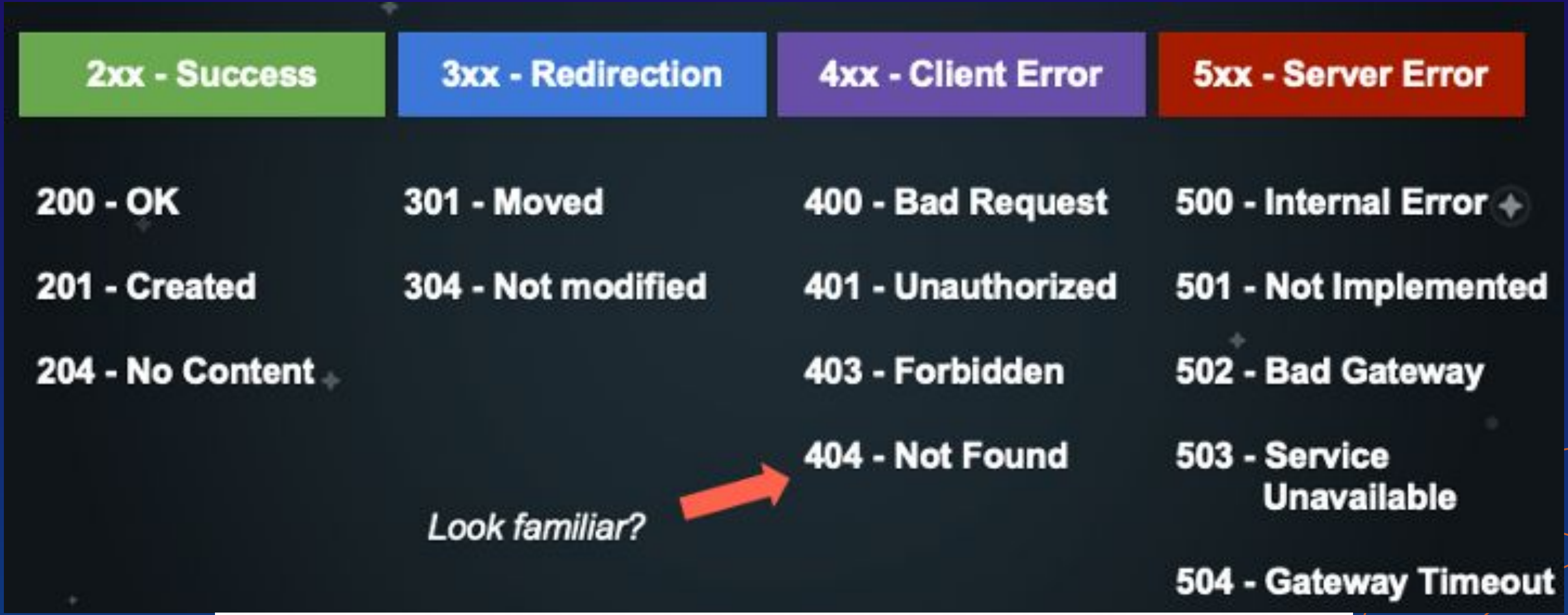
REST Responses

Components of REST Response

- The REST response:

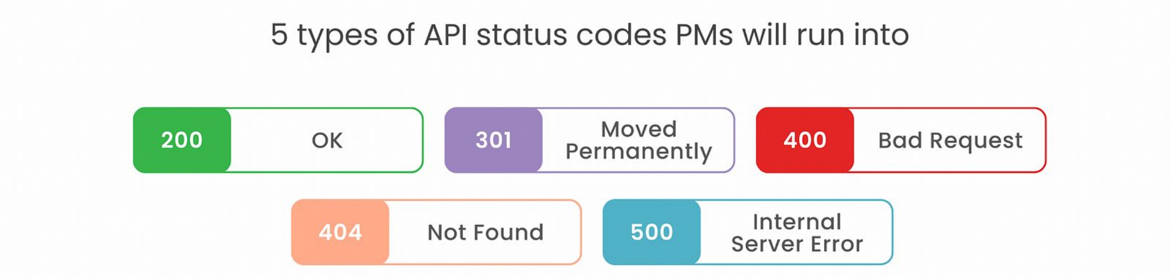
  1. The status code
  2. The headers
  3. The data (or body)

REST Responses: Status Codes



2xx - Success	3xx - Redirection	4xx - Client Error	5xx - Server Error
200 - OK	301 - Moved	400 - Bad Request	500 - Internal Error
201 - Created	304 - Not modified	401 - Unauthorized	501 - Not Implemented
204 - No Content		403 - Forbidden	502 - Bad Gateway
		404 - Not Found	503 - Service Unavailable
			504 - Gateway Timeout

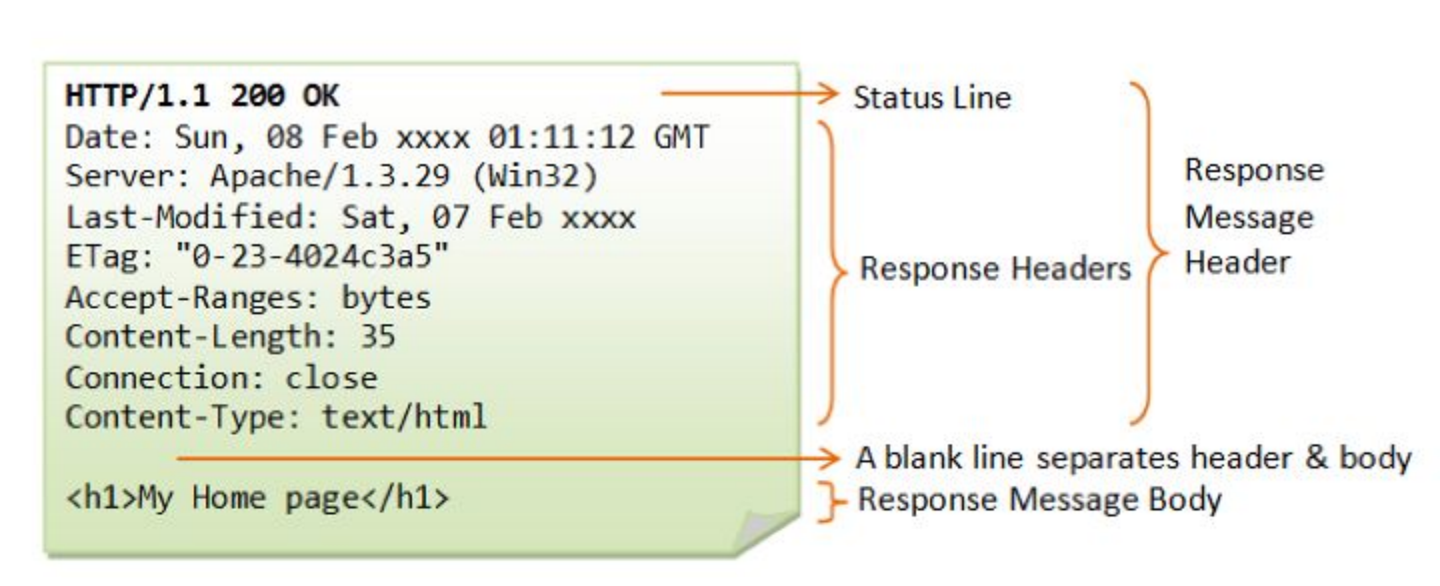
Look familiar?



5 types of API status codes PMs will run into

200	OK
301	Moved Permanently
400	Bad Request
404	Not Found
500	Internal Server Error

REST Response



```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

REST Response

Response Header Example (metadata):

```
Access-Control-Allow-Origin: www.amazon.com
Content-Length: 22
Content-Type: application/json
Date: Mon, 30 Oct 2023 16:32:01 GMT
```

Response Body Example (data):

```
{
  "product_id": 12345,
  "message": "Product 'Smartphone' has been successfully added."
}
```

JSON

JSON

- JSON: JavaScript Object Notation
  - JSON is the standard for transferring data to and from REST APIs
  - REST APIs should accept/send JSON data
  - Despite the name, JSON is a (mostly) language-independent way of specifying objects as name-value pairs

- JSON Objects
  - Look like JavaScript Objects
  - Each property & value must be wrapped with double quotation marks
  - Example:

```
{
  "property1": "value1",
  "property2": "value2",
  "property3": "value3"
}
```

JSON Example

- Example from SecretGeek’s 3 minute JSON tutorial: http://secretgeek.net/json_3mins.asp

```
{"skillz": {
    "web":[
        { "name": "html",
          "years": 5
        },
        { "name": "css",
          "years": 3
        }]
    "database":[
        { "name": "sql",
          "years": 7
        }]
}}
```

JSON Syntax

- An **object** is an unordered set of name/value pairs
  - The pairs are enclosed within braces: `{ }`
  - There is a colon between the name and the value
  - Pairs are separated by commas
  - Example:

```
{ "name": "html", "years": 5 }
```

- An **array** is an ordered collection of values
  - The values are enclosed within brackets: `[ ]`
  - Values are separated by commas
  - Example:

```
[ "html", ”xml", "css"  ]
```

JSON Syntax

- A value can be: A string, a number, true, false, null, an object, or an array
  - Values can be nested

- Strings are enclosed in double quotes
  - Can contain the usual assortment of escaped characters

- Numbers have the usual C/C++/Java syntax
  - Including exponential (E) notation
  - All numbers are decimal--no octal or hexadecimal

- Whitespace can be used between any pair of tokens

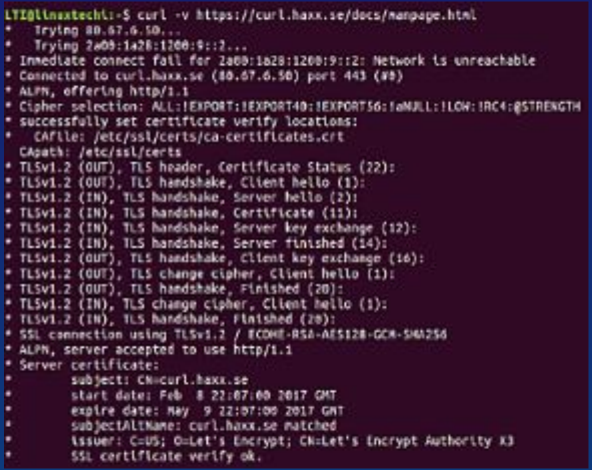
REST in Action

You can send a request with most any programming language!

For class we will use **Postman**, **Python,** and **cURL**

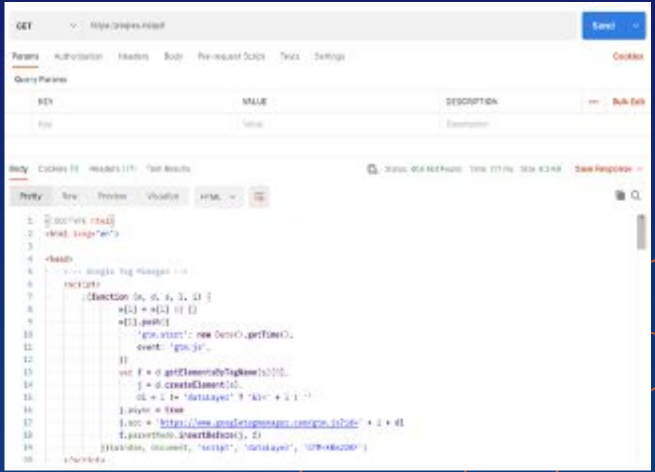
REST in Action

cURL command line



```
LTI@linuxtechi:~$ curl -v https://curl.haxx.se/docs/manpage.html
*   Trying 80.67.6.50...
*   Trying 2a00:1a28:1200:9::2...
* Immediate connect fail for 2a00:1a28:1200:9::2: Network is unreachable
* Connected to curl.haxx.se (80.67.6.50) port 443 (#0)
* ALPN, offering http/1.1
* Cipher selection: ALL:!EXPORT:!EXPORT40:!EXPORT56:!aNULL:!LOW:!RC4:@STRENGTH
* successfully set certificate verify locations:
*   CAfile: /etc/ssl/certs/ca-certificates.crt
  CApath: /etc/ssl/certs
* TLSv1.2 (OUT), TLS header, Certificate Status (22):
* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Client hello (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS change cipher, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES128-GCM-SHA256
* ALPN, server accepted to use http/1.1
* Server certificate:
*        subject: CN=curl.haxx.se
*        start date: Feb  8 22:07:00 2017 GMT
*        expire date: May  9 22:07:00 2017 GMT
*        subjectAltName: curl.haxx.se matched
*        issuer: C=US; O=Let's Encrypt; CN=Let's Encrypt Authority X3
*        SSL certificate verify ok.
```

Postman



What is Postman?

- Postman is a collaborative API development platform that simplifies creating, using, and testing APIs with a UI

- More than 500,000 organizations & 13 Million developers use Postman

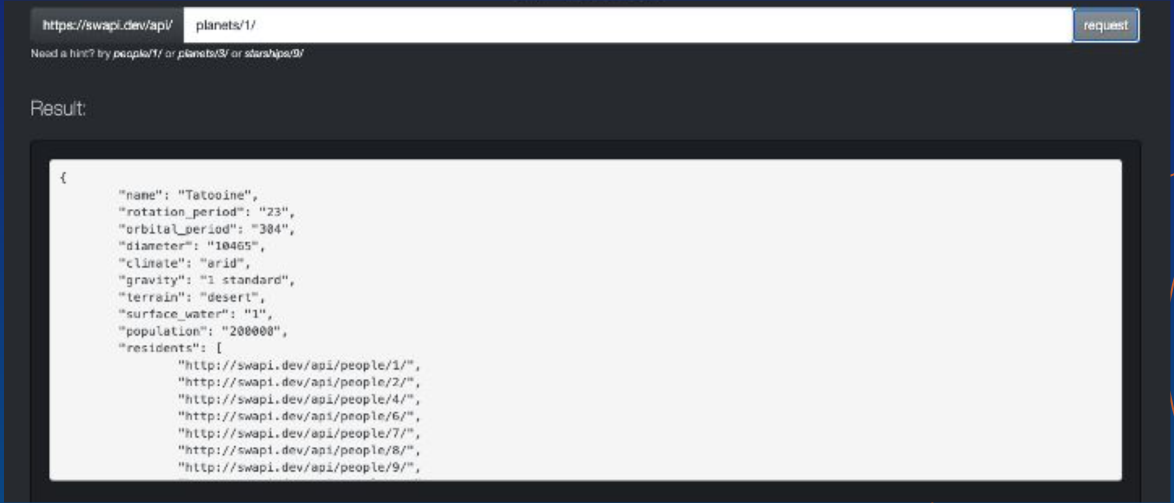
98% of Fortune 500 Organizations use Postman

65% of Global 2000 Organizations use Postman

What is cURL?

- cURL: command line URL tools
    - To install cURL use this link: install curl
    - To use cURL, you type curl in the terminal followed by the endpoint you’re requesting for

- Example: Star Wars API
    - try it now: `curl https://swapi.dev/api/people/1/`



Need a hint? try people/1/ or planets/3/ or starships/9/

Result:

```
{
        "name": "Tatooine",
        "rotation_period": "23",
        "orbital_period": "304",
        "diameter": "10465",
        "climate": "arid",
        "gravity": "1 standard",
        "terrain": "desert",
        "surface_water": "1",
        "population": "200000",
        "residents": [
                "http://swapi.dev/api/people/1/",
                "http://swapi.dev/api/people/2/",
                "http://swapi.dev/api/people/4/",
                "http://swapi.dev/api/people/6/",
                "http://swapi.dev/api/people/7/",
                "http://swapi.dev/api/people/8/",
                "http://swapi.dev/api/people/9/",
```

Up next…

- Sprint 0 Review - due Thursday 2/13
  - Sprint 0 Teamwork Survey due Friday 2/14
- Become a Postman API Student Expert - due Monday Feb 17

Upcoming:

- Understanding APIs Assignment

Sources

https://martinfowler.com/articles/microservices.html

https://medium.com/@akmuthumala/software-architecture-patterns-9e348eb73921

https://www.skiplevel.co/blog/part-2-rest-api-components-how-to-read-them