REST and JSON
JSON
JavaScript Object Notation

• Goal: Transfer data
  • Between computers / processes
  • Between programming languages
• Best for “data” object.
  • Key/Value pairs. Dictionaries, Arrays
• Not great for object relationships
  • Linked Lists, Graphs, OOP
JSON vs XML

fight!

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cas:serviceResponse xmlns:cas="http://www.yale.edu/tp/cas">
  <cas:authenticationSuccess>
    <cas:user>fischerm</cas:user>
    <cas:attributes>
      <cas:dbkey>1111111111111</cas:dbkey>
      <cas:emplid>22222222</cas:emplid>
      <cas:activestudent>0</cas:activestudent>
      <cas:activeemployee>1</cas:activeemployee>
    </cas:attributes>
  </cas:authenticationSuccess>
</cas:serviceResponse>
```

```json
{
  "serviceResponse": {
    "authenticationSuccess": {
      "user": "fischerm",
      "attributes": {
        "dbkey": 1111111111111,
        "emplid": 22222222,
        "activestudent": 0,
        "activeemployee": 1
      }
    }
  }
}
```
JSON vs XML

XML Strengths
- Plain Text
- Flexible
- Can be defined with formal document definitions
- Excellent validation tools

XML Drawbacks
- Verbose
- “Automatic” ideals never fully realized
- Flexibility leads to ambiguous mapping to internal data objects
JSON vs XML

- JSON Strengths
  - Plain Text
  - Literally JavaScript Object Notation
  - Lighter weight formatting. Simple Key/Value
  - Arrays explicitly supported. Allows for cleaner mapping to programming language objects

- JSON Drawbacks
  - No Comments😢
  - Validation was an afterthought
  - Can’t encode complex relationships

```
"serviceResponse": {
  "authenticationSuccess": {
    "user": "fischerm",
    "attributes": {
      "dbkey": 1111111111111,
      "emplid": 22222222,
      "activestudent": 0,
      "activeemployee": 1
    }
  }
}
```
JSON

Formal Grammar

- Parent construct is either an object or array
  - `{ ... }` for Object
  - `[ ... ]` for Array


```json
{  
  "name": "Mark",
  "netid": "fischerm"
}
```

```json
[
  {  
    "name": "Mark",
    "netid": "fischerm"
  },
  {  
    "name": "Yingjie",
    "netid": "ym1014"
  },
  {  
    "name": "Harshita",
    "netid": "harshitanarnoli"
  }
]
```
JSON
Formal Grammar

• Whitespace is unimportant

• These are all equivalent
JSON

Objects

• Objects are defined with curly braces
• Key/Value pairs are separated by a colon
• Keys are strings
• Values can be strings, numbers, boolean, null, objects, or arrays
• "key": "value" pairs separated by commas
• Trailing commas are not allowed

{  
  "name": "Mark",
  "netid": "fischerm"
}

{  
  "name": "Mark",
  "netid": "fischerm",
}


JSON

Objects

• Keys must be strings

• Double Quotes are required

```json
{
    "name": "Mark",
    "netid": "fischerm"
}
```

```json
{
    'name': 'Mark',
    'netid': 'fischerm'
}
```

JSON

Arrays

• Arrays are defined by square brackets

• Comma separated list of values

• Values can be strings, numbers, boolean, null, objects, or arrays
JSON For Humans

• Many code editors will auto-format JSON for you

• Postman will pretty-print JSON output or show raw

• Some browsers display pretty-print JSON, others render it as an expandable tree
**JSON**

In JavaScript

- Language level JSON object
- Can’t create instances with `new`, static methods only
- JavaScript → JSON

```javascript
let obj = {
  'books': [
    {
      'title': "There and Back Again",
      'author': "Bilbo Baggins"
    },
    {
      'title': "The Downfall of the Lord of the Rings, and the Return of the King",
      'author': "Frodo Baggins"
    }
  ]
}

console.log(JSON.stringify(obj))
```

`JSON.stringify( var )`
JSON
In JavaScript

• JSON → JavaScript

```javascript
let jsonString = '{"title":"There and Back Again","author":"Bilbo Baggins"}'

book = JSON.parse( jsonString )
console.log( book.title )
```
JSON

In JavaScript

- JSON is always valid JavaScript
- JavaScript is **NOT** always valid JSON

```javascript
let obj = {
  'books': [
    {
      'title': "There and Back Again",
      'author': "Bilbo Baggins"
    },
    {
      'title': "The Downfall of the Lord of the Rings, and the Return of the King",
      'author': "Frodo Baggins"
    }
  ]
}

console.log( JSON.stringify(obj) )
```
In Python

- json module is part of the Python standard library
- JSON → Python

```python
import json
jsonString = '{"title":"There and Back Again","author":"Bilbo Baggins"}'
print(json.loads(jsonString))
```

```bash
~/Demo $ python json_demo.py
{'title': 'There and Back Again', 'author': 'Bilbo Baggins'}
```
In Python

- Python → JSON

```python
import json

obj = {
    "books": [
        {
            "title": "There and Back Again"
        },
        {
            "title": "The Downfall of the Lord of the Rings, and the Return of the King",
            "author": "Frodo Baggins"
        }
    ]
}

print(json.dumps(obj))
```
JSON

In Python

- Python → JSON
- Optional `indent` argument to `dumps` will pretty-print your JSON strings from Python

```python
import json
...
print(json.dumps(obj, indent=2))
```
REST
Representational State Transfer
REST
Representational State Transfer

- JSON objects = DB records
- Send & Receive over HTTP
- URLs = object IDs
REST
GitHub API

• For example here is the GitHub API call to list basic info about my personal GitHub account

GET https://api.github.com/users/estranged42
GET https://api.github.com/

GET https://api.github.com/users/estranged42

```json
{
  "login": "estranged42",
  "id": 2087572,
  "node_id": "MDQ6VXNlcjIwODc1NzI=",
  "gravatar_id": "",
  "url": "https://api.github.com/users/estranged42",
  "html_url": "https://github.com/estranged42",
  "followers_url": "https://api.github.com/users/estranged42/followers",
  "following_url": "https://api.github.com/users/estranged42/following(/other_user)",
  "gists_url": "https://api.github.com/users/estranged42/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/estranged42/starred(/owner){/repo}",
  "subscriptions_url": "https://api.github.com/users/estranged42/subscriptions",
  "organizations_url": "https://api.github.com/users/estranged42/organizations",
  "repos_url": "https://api.github.com/users/estranged42/repos",
  "events_url": "https://api.github.com/users/estranged42/events{/privacy}",
  "received_events_url": "https://api.github.com/users/estranged42/received_events",
  "type": "User",
  "site_admin": false,
  "name": "Mark Fischer",
  "company": null,
  "blog": "http://www.fischco.org",
  "location": null,
  "email": null,
  "hireable": null,
  "bio": "#web101 Podcast: https://web101.org\n\nhttps://keybase.io/estranged",
  "twitter_username": "estranged",
  "public_repos": 23,
  "public_gists": 3,
  "followers": 4,
}
```
REST
GitHub API

- Since I requested a single thing, I received a dictionary in response:
REST
GitHub API

• If I request all of my repositories, I’ll receive an array response

GET https://api.github.com/users/estranged42/repos
REST
GitHub API

- Typically all the records in a list will have the same fields, although JSON does not enforce this.
REST
GitHub API

• Typically records will have some sort of unique identifier
REST
GitHub API

- There are specific URLs for each individual repository

GET https://api.github.com/repos/estranged42/ArduinoCore-samd
REST
Fundamentals

- REST is not a protocol, like HTTP, or SOAP
- REST is an architectural style, defined by a few key principles

https://en.wikipedia.org/wiki/Representational_state_transfer
REST
Client-Server Architecture

• Separation of concerns
• Decouples user interface from data access and persistence
• Allows for many different architectures for client and server
REST
Uniform Interface

- Requests should identify resources
  - They do so by using a uniform resource identifier (URI)
- Resource manipulation through representations
  - When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource's state
- Self-descriptive messages contain metadata about how the client can best use them
- A REST client should then be able to use server-provided links dynamically to discover all the available resources it needs
REST
Statelessness

- Clients can request resources in any order, and every request is stateless or isolated from other requests
- Statelessness refers to a communication method in which the server completes every client request independently of all previous requests
- Implies that the server can completely understand and fulfill the request every time
REST
Layered System

- A client can connect to other authorized intermediaries between the client and server, and it will still receive responses from the server.
- Design your RESTful web service to run on several servers with multiple layers such as security, application, and business logic, working together to fulfill client requests.
- These layers remain invisible to the client.
REST
Cacheability

• As on the World Wide Web, clients and intermediaries can cache responses
• Well-managed caching partially or completely eliminates some client–server interactions, further improving scalability and performance
• The cache can be performed at the client machine in memory or browser cache storage
• Additionally cache can be stored in a Content Delivery Network (CDN)
## REST

### Semantic HTTP Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Get a representation of the target resource’s state</td>
</tr>
<tr>
<td>POST</td>
<td>Let the host process a resource state sent in the request</td>
</tr>
<tr>
<td>PUT</td>
<td>Create or replace the state of a target resource with the state defined in the request</td>
</tr>
<tr>
<td>PATCH</td>
<td>Partially update a resource’s state</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete the target resource’s state</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Describe the available methods</td>
</tr>
</tbody>
</table>