Week 5:
Data Management and Integration, Databases and Application Programming Interfaces

Javascript Object Notation (JSON)

JOSEPH BENJAMIN R. ILAGAN

What is JSON?

In computing, JavaScript Object Notation (**JSON**) (/ˈdʒeɪsən/ "Jason",[1] /ˈdʒeɪsɒn/) is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value). It is a very common data format used for asynchronous browser–server communication, including as a replacement for XML in some AJAX-style systems.[2]

What is JSON?

JSON is a language-independent data format. It was derived from JavaScript, but as of 2017, many programming languages include code to generate and parse JSON-format data. The official Internet media type for JSON is application/json. JSON filenames use the extension .json.

Source: Wikipedia

Sample JSON String

```
{"code":"espresso",
 "name":"Espresso",
 "price":"140"}
```

Sample JSON String


```
[{"code":"espresso","name":"Espresso","price
":"140"},
{"code":"americano","name":"Americano","pric
e":"150"},
{"code":"cappuccino","name":"Cappuccino","pr
ice":"170"}
]
```

Note: Regular strings are JSON too

```
The quick brown fox
```

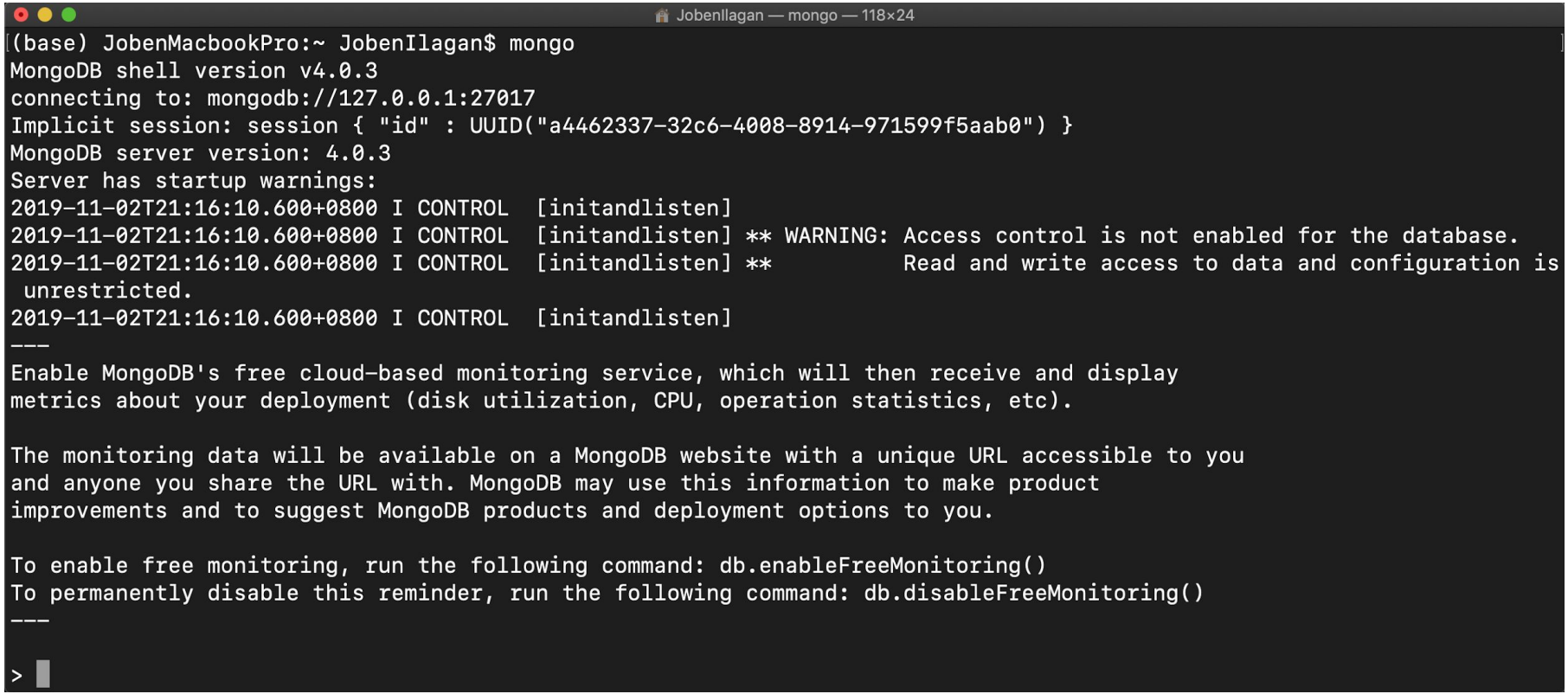
Processing JSON in Python

Database Management using NoSQL

JOSEPH BENJAMIN R. ILAGAN

Download and setup MongoDB on your Mac or Windows machines

Run the MongoDB Shell



Creating databases

Creating our first few databases

We will create three initial databases for our fictitious retail business:

`products`: stores product information including prices and current inventory quantities

`order_management` : stores customer sales order transactions

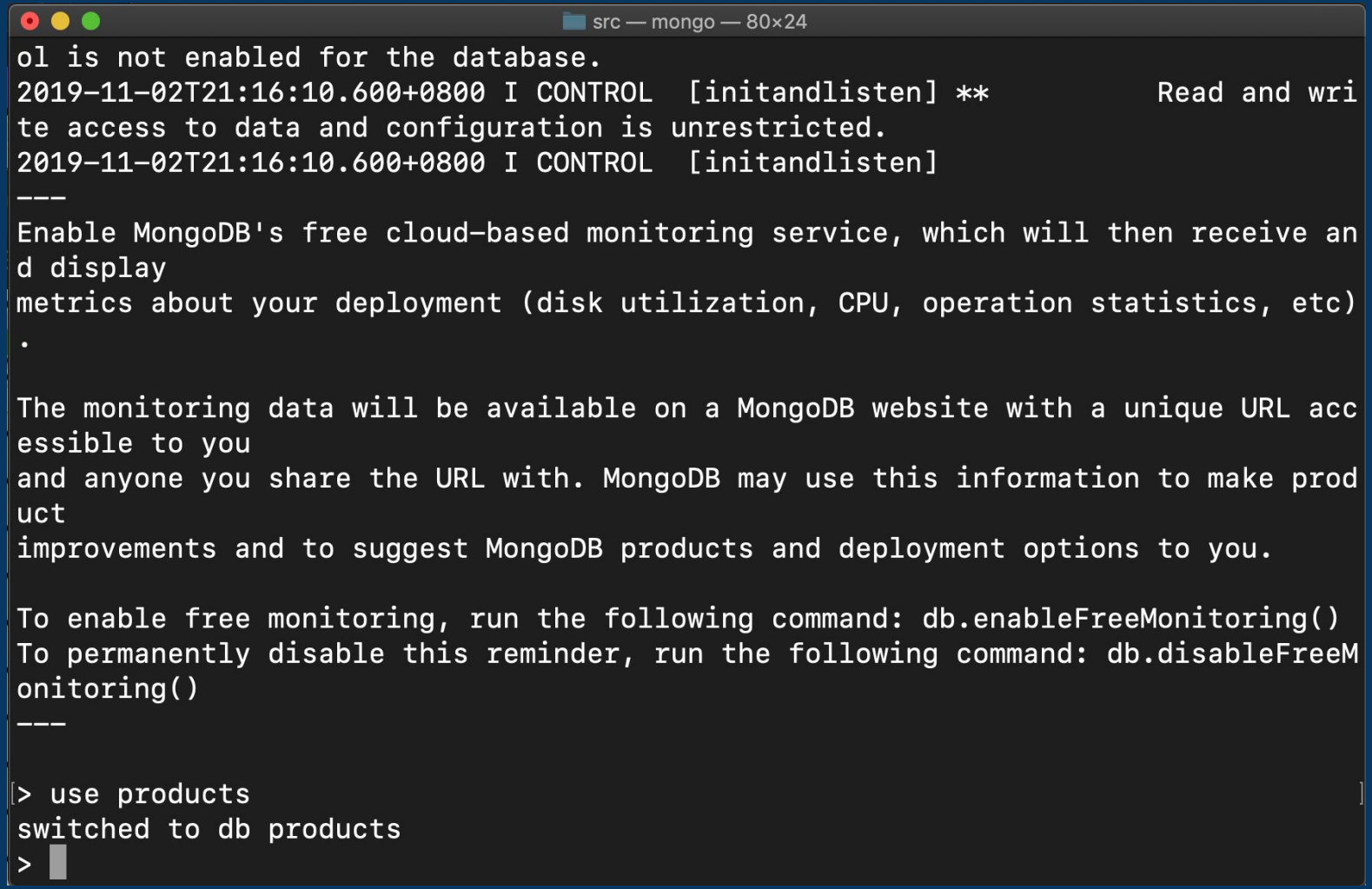
`hr`: stores employee information, including salaries and employment status

Creating our first few databases

If you want to use a database with name **<mydb>**, then use DATABASE statement would be as follows −

```
> use products
switched to db products
```

In reality, MongoDB does not create the databases until content is made available.


src — mongo — 80×24
ol is not enabled for the database.
2019-11-02T21:16:10.600+0800 I CONTROL [initandlisten] ** Read and wri
te access to data and configuration is unrestricted.
2019-11-02T21:16:10.600+0800 I CONTROL [initandlisten]

Enable MongoDB's free cloud-based monitoring service, which will then receive an
d display
metrics about your deployment (disk utilization, CPU, operation statistics, etc)
.
The monitoring data will be available on a MongoDB website with a unique URL acc
essible to you
and anyone you share the URL with. MongoDB may use this information to make prod
uct
improvements and to suggest MongoDB products and deployment options to you.
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeM
onitoring()

> use products
switched to db products
>

Creating collections and documents

Creating our first collections and documents

Following is an example where we shall try creating a collection named `customers` in the `order_management` database. (DO NOT type the ">" character.)

```
> use order_management
Switched to db order_management
```

```
> db.customers.insert({ "first_name":
"Rodrigo", "last_name": "Duterte", "city":
"Manila", "points":1000 })
```

Inserting multiple documents

```
db.customers.insertMany([
{"username":"chums@example.com",
 "password":"Ch@ng3m3!",
 "first_name":"Matthew",
 "last_name":"Uy"
},
{"username":"joben@example.com",
 "password":"Ch@ng3m3!",
 "first_name":"Joben",
 "last_name":"Ilagan"
},
]);
```

Inserting a few more

```
db.customers.insertMany([
{"username":"bong@example.com",
 "password":"Ch@ng3m3!",
 "first_name":"Bong",
 "last_name":"Olpoc"
},
{"username":"joaqs@example.com",
 "password":"Ch@ng3m3!",
 "first_name":"Joaqs",
 "last_name":"Gonzales"
},
]);
```

Inserting a few more

```
db.customers.insertMany([
{"username":"gihoe@example.com",
 "password":"Ch@ng3m3!",
 "first_name":"Gio",
 "last_name":"Hernandez"
},
{"username":"vic@example.com",
 "password":"Ch@ng3m3!",
 "first_name":"Vic",
 "last_name":"Reventar"
},
]);
```

Inserting a few more

```
db.customers.insertMany([
{"username":"joe@example.com",
 "password":"Ch@ng3m3!",
 "first_name":"Joe",
 "last_name":"Ilagan"
},
{"username":"eb@example.com",
 "password":"Ch@ng3m3!",
 "first_name":"EB",
 "last_name":"Ilagan"
},
]);
```

Insert Product Data

```
> use products
switched to db products

> db.products.insertMany([
    {"code":NumberInt(100),"name":"Americano","price":125},
    {"code":NumberInt(200),"name":"Brewed Coffee","price":110},
    {"code":NumberInt(300),"name":"Cappuccino","price":120},
    {"code":NumberInt(400),"name":"Espresso","price":120},
    {"code":NumberInt(500),"name":"Latte","price":140},
    {"code":NumberInt(600),"name":"Cold Brew","price":200}
]);
```

Results (do not copy/execute)

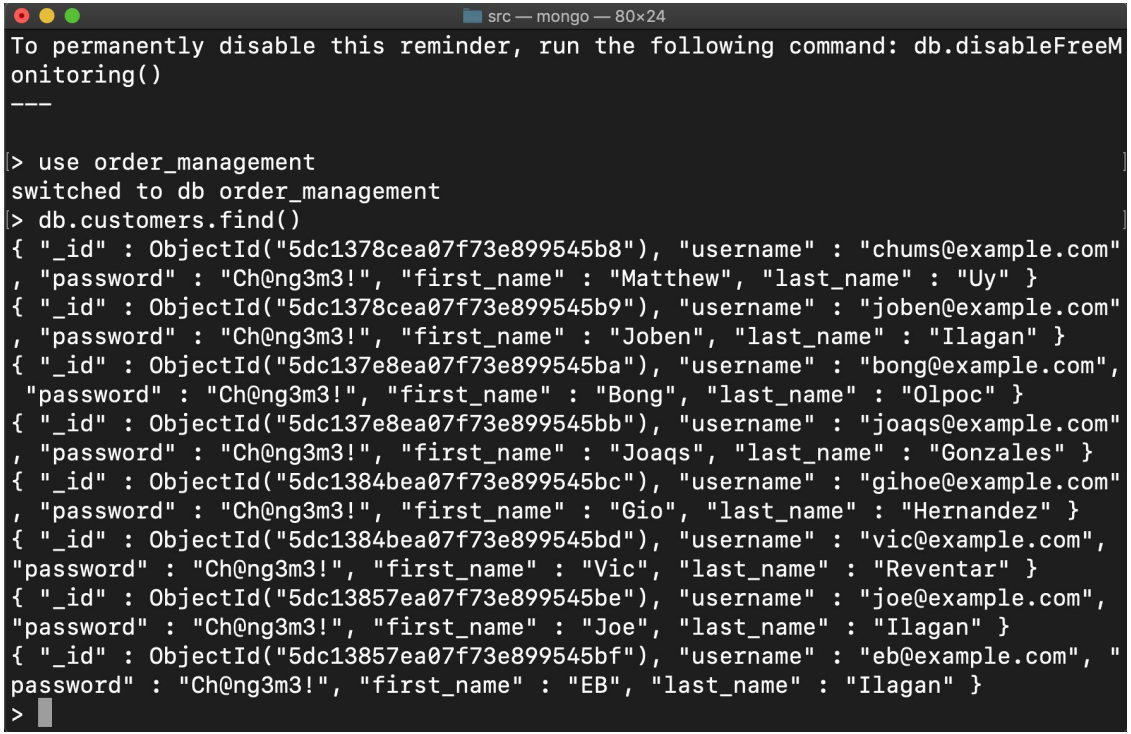
```
{
    "acknowledged" : true,
    "insertedIds" : [
        ObjectId("5dcb7321cedc1b3b09db7bc3"),
        ObjectId("5dcb7321cedc1b3b09db7bc4"),
        ObjectId("5dcb7321cedc1b3b09db7bc5"),
        ObjectId("5dcb7321cedc1b3b09db7bc6"),
        ObjectId("5dcb7321cedc1b3b09db7bc7"),
        ObjectId("5dcb7321cedc1b3b09db7bc8")
    ]
}
```

Show Collections

```
> show collections
products
```

Queries

Query all


src — mongo — 80×24
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()

> use order_management
switched to db order_management
> db.customers.find()
{ "_id" : ObjectId("5dc1378cea07f73e899545b8"), "username" : "chums@example.com", "password" : "Ch@ng3m3!", "first_name" : "Matthew", "last_name" : "Uy" }
{ "_id" : ObjectId("5dc1378cea07f73e899545b9"), "username" : "joben@example.com", "password" : "Ch@ng3m3!", "first_name" : "Joben", "last_name" : "Ilagan" }
{ "_id" : ObjectId("5dc137e8ea07f73e899545ba"), "username" : "bong@example.com", "password" : "Ch@ng3m3!", "first_name" : "Bong", "last_name" : "Olpoc" }
{ "_id" : ObjectId("5dc137e8ea07f73e899545bb"), "username" : "joaqs@example.com", "password" : "Ch@ng3m3!", "first_name" : "Joaqs", "last_name" : "Gonzales" }
{ "_id" : ObjectId("5dc1384bea07f73e899545bc"), "username" : "gihoe@example.com", "password" : "Ch@ng3m3!", "first_name" : "Gio", "last_name" : "Hernandez" }
{ "_id" : ObjectId("5dc1384bea07f73e899545bd"), "username" : "vic@example.com", "password" : "Ch@ng3m3!", "first_name" : "Vic", "last_name" : "Reventar" }
{ "_id" : ObjectId("5dc13857ea07f73e899545be"), "username" : "joe@example.com", "password" : "Ch@ng3m3!", "first_name" : "Joe", "last_name" : "Ilagan" }
{ "_id" : ObjectId("5dc13857ea07f73e899545bf"), "username" : "eb@example.com", "password" : "Ch@ng3m3!", "first_name" : "EB", "last_name" : "Ilagan" }
>

Query All

```
> use order_management
switched to db order_management
> show collections
customers
> db.customers.find()
```

```
{ "_id" : ObjectId("5dc1378cea07f73e899545b8"), "username" : "chums@example.com",
"password" : "Ch@ng3m3!", "first_name" : "Matthew", "last_name" : "Uy" }
{ "_id" : ObjectId("5dc1378cea07f73e899545b9"), "username" : "joben@example.com",
"password" : "Ch@ng3m3!", "first_name" : "Joben", "last_name" : "Ilagan" }
{ "_id" : ObjectId("5dc137e8ea07f73e899545ba"), "username" : "bong@example.com",
"password" : "Ch@ng3m3!", "first_name" : "Bong", "last_name" : "Olpoc" }
{ "_id" : ObjectId("5dc137e8ea07f73e899545bb"), "username" : "joaqs@example.com",
"password" : "Ch@ng3m3!", "first_name" : "Joaqs", "last_name" : "Gonzales" }
{ "_id" : ObjectId("5dc1384bea07f73e899545bc"), "username" : "gihoe@example.com",
"password" : "Ch@ng3m3!", "first_name" : "Gio", "last_name" : "Hernandez" }
{ "_id" : ObjectId("5dc1384bea07f73e899545bd"), "username" : "vic@example.com",
"password" : "Ch@ng3m3!", "first_name" : "Vic", "last_name" : "Reventar" }
{ "_id" : ObjectId("5dc13857ea07f73e899545be"), "username" : "joe@example.com",
"password" : "Ch@ng3m3!", "first_name" : "Joe", "last_name" : "Ilagan" }
{ "_id" : ObjectId("5dc13857ea07f73e899545bf"), "username" : "eb@example.com",
"password" : "Ch@ng3m3!", "first_name" : "EB", "last_name" : "Ilagan" }
```

Query Several

```
> db.customers.find({"last_name":"Ilagan"})
{ "_id" : ObjectId("5dc1378cea07f73e899545b9"),
"username" : "joben@example.com", "password" :
"Ch@ng3m3!", "first_name" : "Joben", "last_name" :
"Ilagan" }
{ "_id" : ObjectId("5dc13857ea07f73e899545be"),
"username" : "joe@example.com", "password" :
"Ch@ng3m3!", "first_name" : "Joe", "last_name" :
"Ilagan" }
{ "_id" : ObjectId("5dc13857ea07f73e899545bf"),
"username" : "eb@example.com", "password" : "Ch@ng3m3!",
"first_name" : "EB", "last_name" : "Ilagan" }
```

Query One

```
> db.customers.findOne({"last_name":"Ilagan"})
{
    "_id" : ObjectId("5dc1378cea07f73e899545b9"),
    "username" : "joben@example.com",
    "password" : "Ch@ng3m3!",
    "first_name" : "Joben",
    "last_name" : "Ilagan"
}
```

Query One

```
> db.customers.findOne("5dc1378cea07f73e899545b9")
{
    "_id" : ObjectId("5dc1378cea07f73e899545b9"),
    "username" : "joben@example.com",
    "password" : "Ch@ng3m3!",
    "first_name" : "Joben",
    "last_name" : "Ilagan"
}
```

Query One

```
>
db.customers.findOne({"_id":ObjectId("5dc1378cea07f73e89
9545b9")})
{
    "_id" : ObjectId("5dc1378cea07f73e899545b9"),
    "username" : "joben@example.com",
    "password" : "Ch@ng3m3!",
    "first_name" : "Joben",
    "last_name" : "Ilagan"
}
```

Query All

```
> db.products.find()
```

Let's add transaction data

Updates

Update Functions

- `replaceOne()`
- `updateOne()`
- `updateMany()`

Update One (copy the command in bold)

```
> use order_management
switched to db order_management
> show collections
customers
> db.customers.find({"username":"chums@example.com"},{"password":1})
{ "_id" : ObjectId("5dc1378cea07f73e899545b8"), "password" :
"Ch@ng3m3!" }
> db.customers.updateOne({"username":"chums@example.com"},
{"$set":{"password":"n3wp@ssw0rd"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 0 }
> db.customers.find({"username":"chums@example.com"},
{"_id":0,"password":1})
{ "password" : "n3wp@ssw0rd" }
```

Update Many

```
> db.products.updateMany({},{"$mul":{"price":1.2}})
```

Accessing MongoDB from Python

Application Programming Interfaces (APIs)

JOSEPH BENJAMIN R. ILAGAN

What is an API?

An application programming interface (API) is an interface or communication protocol between a client and a server intended to simplify the building of client-side software. It has been described as a “contract” between the client and the server, such that if the client makes a request in a specific format, it will always get a response in a specific format or initiate a defined action.

Source: Wikipedia

What is an API?

An API may be for a web-based system, operating system, database system, computer hardware, or software library.

What is an API?

An API specification can take many forms, but often includes specifications for routines, data structures, object classes, variables, or remote calls.

Documentation for the API usually is provided to facilitate usage and implementation.

What is Web API?

A Web API is an application programming interface for either a web server or a web browser.

Web Services

In a Web service, a Web technology such as HTTP — originally designed for human-to-machine communication — is used for transferring machine-readable file formats such as XML and JSON.

REST

Representational state transfer (REST) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called RESTful Web services, provide interoperability between computer systems on the Internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations. Other kinds of Web services, such as SOAP Web services, expose their own arbitrary sets of operations.

REST

Representational state transfer (REST) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called RESTful Web services, provide interoperability between computer systems on the Internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations. Other kinds of Web services, such as SOAP Web services, expose their own arbitrary sets of operations.

HTTP Methods (Expanded)

HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/ Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/ Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

Resource Naming

In addition to utilizing the HTTP verbs appropriately, resource naming is arguably the most debated and most important concept to grasp when creating an understandable, easily leveraged Web service API. When resources are named well, an API is intuitive and easy to use. Done poorly, that same API can feel klutzy and be difficult to use and understand. Below are a few tips to get you going when creating the resource URIs for your new API.

Source: RESTful Resource Naming, from REST API Tutorial (https://www.restapitutorial.com/lessons/restfulresourcenaming.html)

Example Resources

- Users of the system.
- Courses in which a student is enrolled.
- A user's timeline of posts.
- The users that follow another user.
- An article about horseback riding.

HTTP Status Codes

- **1xx Informational** – the request was received, continuing process
- **2xx Successful** – the request was successfully received, understood and accepted
- **3xx Redirection** – further action needs to be taken in order to complete the request
- **4xx Client Error** – the request contains bad syntax or cannot be fulfilled
- **5xx Server Error** – the server failed to fulfill an apparently valid request