API Development

CPE405 - Advanced Computer Engineering Technology

Dome Potikanond
Navadon Khunlertgit
Banana Software co., Ltd

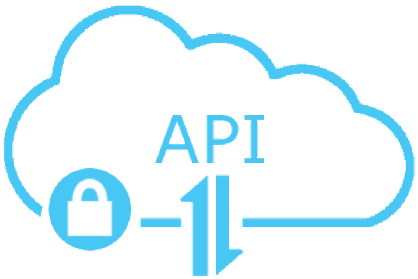
Computer Engineering, Chiang Mai University

Agenda

- What is API?
- API Architecture
- HTML respond
- JSON
- GraphQL

What is an API?

- Application Program Interface
  - Web APIs = Web services
- APIs are everywhere

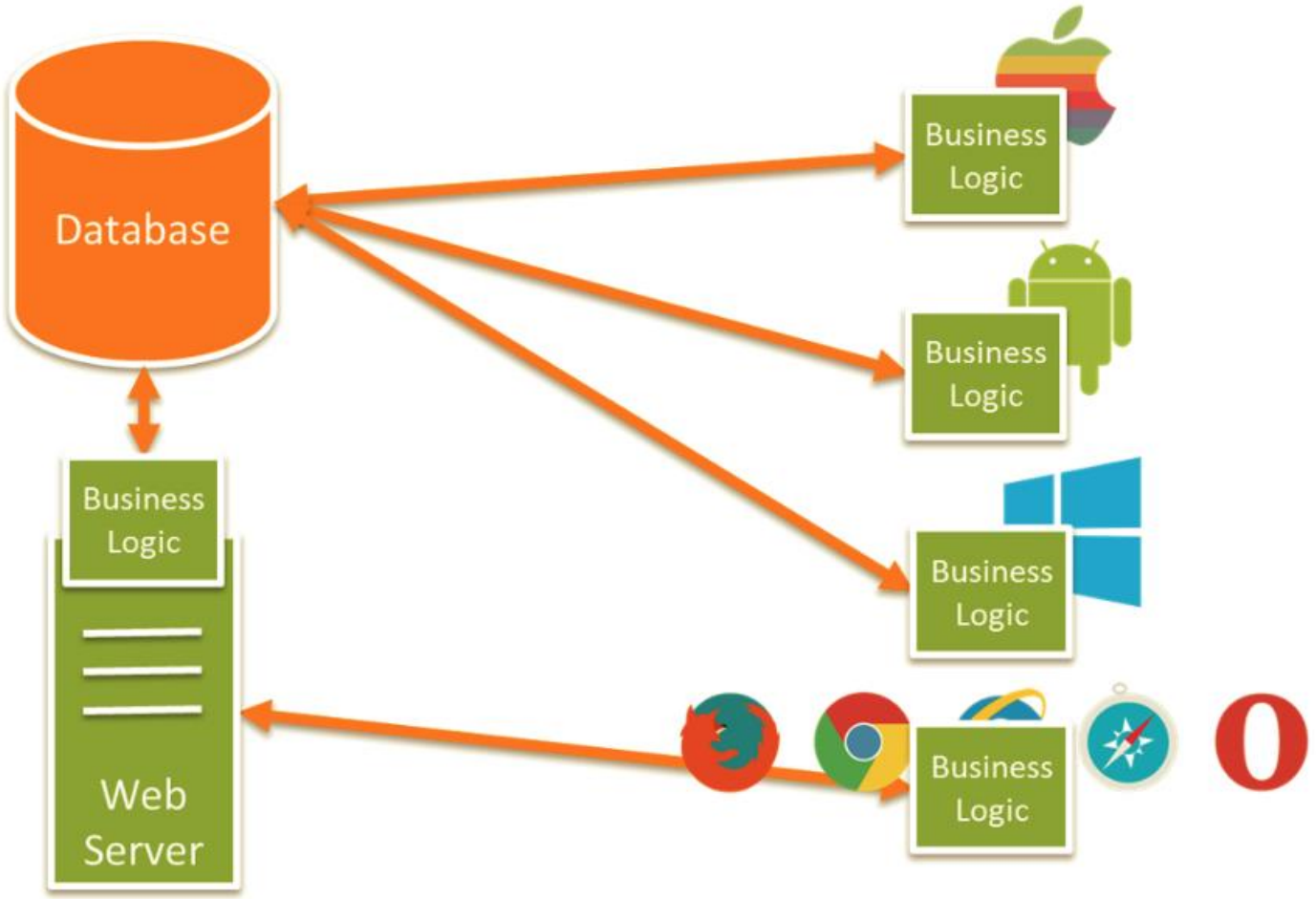


- Contract provided by one piece of software to another
  - A messenger between running software
- Structured request and response
  - **JSON** – JavaScript Object Notation
- SOAP vs. REST
  - 70% of public APIs is REST

{ json:api }

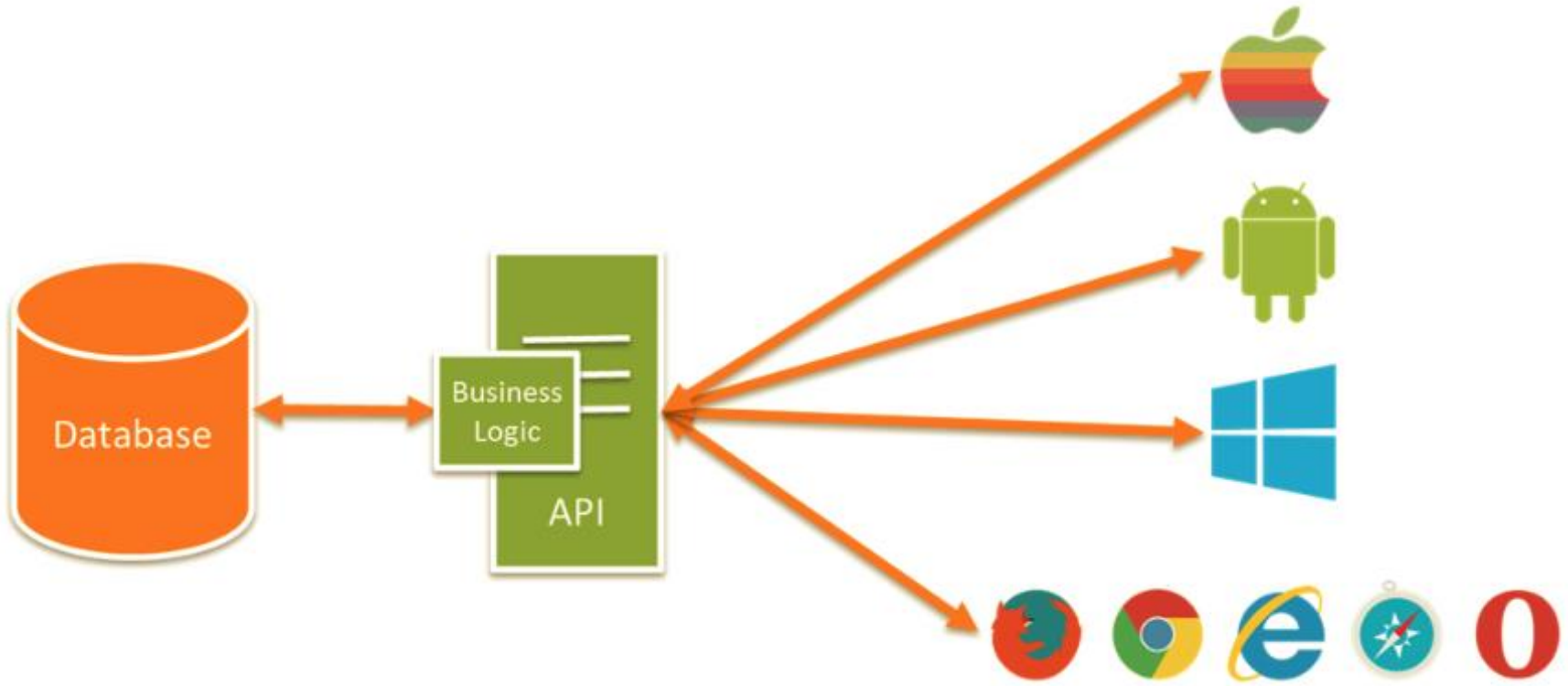
Why API?

- Each client app has its own embedded business logic



Central API

- Each app uses the **same API** to get and manipulate data



https://blogs.msdn.microsoft.com/martinkearn/2015/01/05/introduction-to-rest-and-net-web-api/

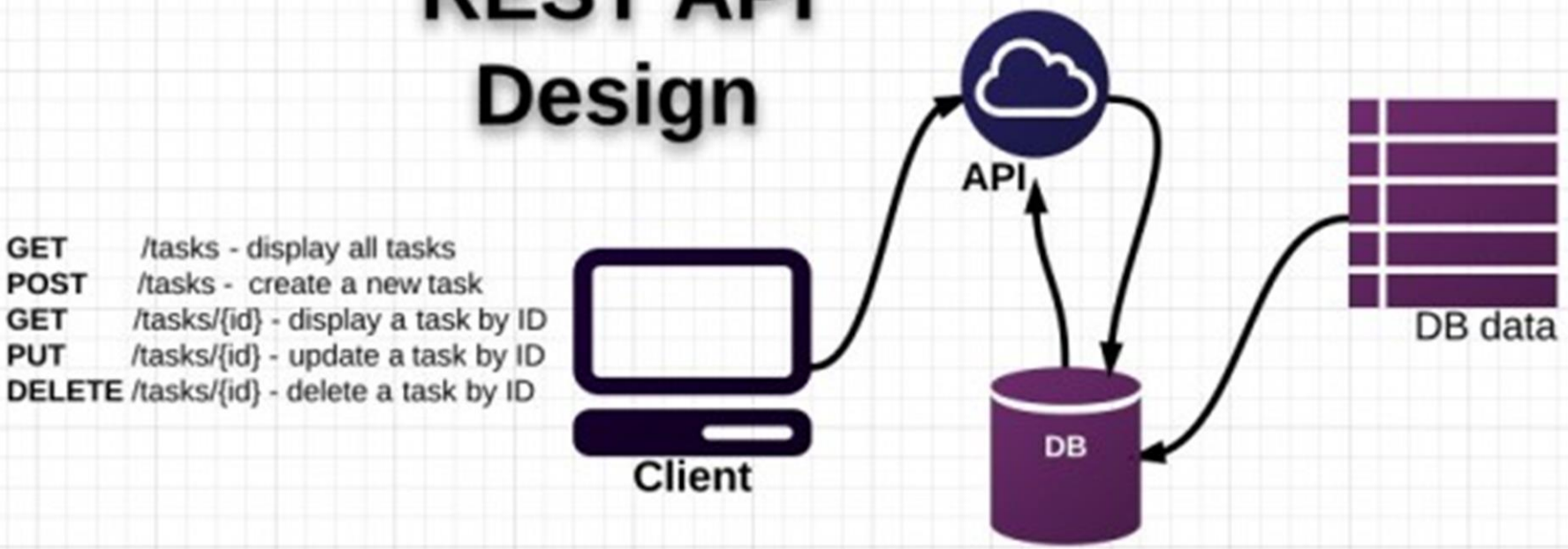
{ REST }

What is a REST API?

What is REST?

- Representational State Transfer
- Architecture style for designing networked applications
- Relies on a stateless, client-sever protocol, almost always HTTP
- Treats server objects as resources that can be created, updated or delete
- Can be used by virtually any programming language

REST API Design



HTTP Methods

- **GET** – Retrieve data from a specified resource
  - The most common
- **POST** – Submit data to be processed to a specified resource
  - Filling a web form (using form action)
- **PUT** – Update a specified resource
  - Send a request to an endpoint, URI, with **id** of a resource
  - Cannot make PUT request from a form
  - Uses AJAX, JavaScript or JQuery to make this request
- **DELETE** – Delete a specified resource

HTTP Methods (cont.)

Other type or requests but rarely used

- **HEAD** – Same as GET but does not return a body
  - Only returns the header info
- **OPTIONS** – Returns the supported HTTP methods
- **PATCH** – Update partial resources

Endpoints

The **URI/URL** where api/service can be accessed by client

- **GET** https://mysite.com/api/users
- **GET** https://mysite.com/api/users/:id
  or https://mysite.com/api/users/details/:id
- **POST** https://mysite.com/api/users
- **PUT** https://mysite.com/api/users/:id
  or https://mysite.com/api/users/update/:id
- **DELETE** https://mysite.com/api/users/:id
  or https://mysite.com/api/users/delete/:id

Endpoints (cont.)

Resource	Verb	Expected Outcome	Response Code
/Products	GET	A list of all products in the system	200/OK
/Products?Colour=red	GET	A list of all products in the system where the colour is red	200/OK
/Products	POST	Creation of a new product	201/Created
/Products/81	GET	Product with ID of 81	200/OK
/Products/881(a product ID which does not exist)	GET	Some error message	404/Not Found
/Products/81	PUT	An update to the product with an ID of 81	204/No Content
/Products/81	DELETE	Deletion of the product with an ID of 81	204/No Content
/Customers	GET	A list of all customers	200/OK

RESTful API

GET PUT POST DELETE

Authentication

Some API's require authentication to use their service

Authentication Options

- **OAuth2** – get a token and sends it with the requests

Example:

- # curl –H "Authorization: token OAUTH-TOKEN" https://api.github.com
- # curl https://api.github.com/?access_token=OAUTH-TOKEN
- # curl "https://api.github.com/users/whatever?client_id=xxx
  &client_secret=yyy"

GitHub API example

- **REST API v3**
  - https://developer.github.com/v3/
  - Explain endpoints & methods you need to use
- Authentication methods
  - https://github.com/settings/applications/new
- **HTTP Verbs** – list of possible types of request
- **Pagination** – specifies # items per page
- Example:
  - GET https://api.github.com/users
  - GET https://api.github.com/users/:username

Resources & Tools

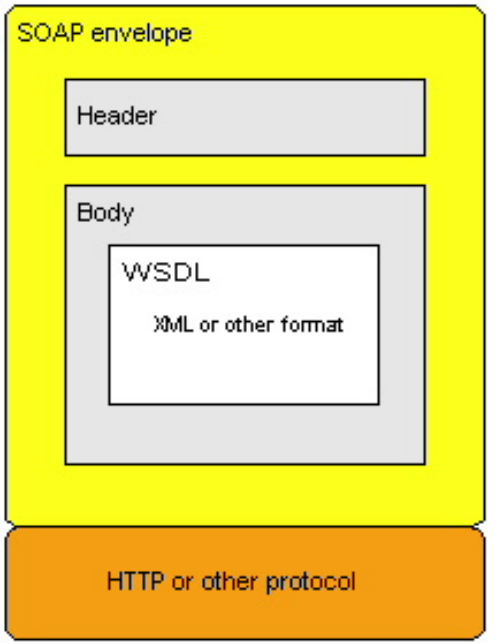
- **Postman** app or chrome extension
  - https://www.getpostman.com/

- Fake Online REST API for Testing
  - https://jsonplaceholder.typicode.com/
    - Posts, comments, albums, photos, todos, users
  - http://fakerestapi.azurewebsites.net/
    - Activities, authors, books, coverphotos, users
  - https://reqres.in/
    - Users (with many endpoints)

SOAP Architecture

SOAP API

- Simple Object Access Protocol
  - https://www.w3schools.com/xml/xml_soap.asp
  - Has been around since 1990
- A **SOAP** message is an XML document containing many elements
  - An envelope – identifies as SOAP message
  - A header – contains header info
  - A body – contains call and response info
  - A fault – contains errors and status info
- Very complex and not very readable



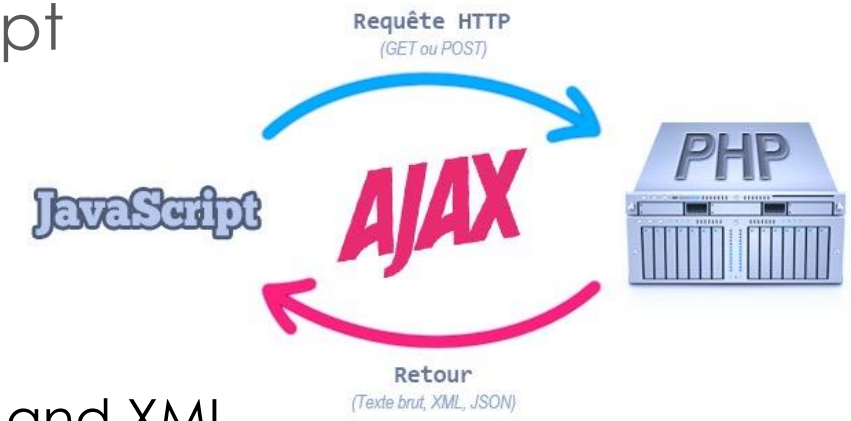
SOAP vs. REST

SOAP	REST
An XML-based message protocol	An architectural style protocol
Uses WSDL for communication	Uses JSON to send and receive data
Invoke services by calling RPC method	Simply calls service via URL path
Result is not easily readable	Result is readable (raw XML or JSON)
Transfer is over HTTP, SMTP, FTP, etc.	Transfer is over HTTP only
Difficult to call SOAP using JavaScript	Easy to call from JavaScript
Performance is not as good as REST	Less CPU intensive than SOAP, cleaner code


{JSON}
JavaScript Object Notation

What is JSON?

- JavaScript Object Notation
- Lightweight data-interchange format
- Based on a subset of JavaScript
- Easy to read and write
- Often used with AJAX
  - **AJAX** – Asynchronous JavaScript and XML
- Can be used with most modern languages



JSON Data Types

- Number – No difference between integer and floats
- String – String of Unicode characters. Use double quotes
- Boolean – True or false
- Array: Ordered list of 0 or more values
- Object: Unordered collection of key/value pairs
- Null – Empty value

JSON Syntax Rules

- Uses key/value pairs – {“name”:”Yuthapong”}
- Uses double quotes around KEY and VALUE
- Must use the specified data types
- File type is “**.json**”
  - Must be valid JSON
- **MIME** type is “**application/json**”
  - Uses for Restful API

JSON Example

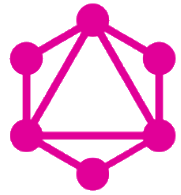
```
{
        “name” : ”Yuthaporn Spears”,
        “age” : 35,
        “address” : {
                “street” : ”123 Chiang Mai – Hang Dong”,
                “province” : ”Chiang Mai”,
                “country” : “Thailand”
        },
        “children” : [“Britney”,”Rihannha”]
}
```

Resources

- **XML HttpRequest**
  - All modern web browser have a built-in XMLHttpRequest object to request data from a server.
  - Update a web page without reloading the page
    - Request data from a server
    - Receive data from a server
    - Send data to server in the background
  - https://www.w3schools.com/xml/xml_http.asp

Resources

- **JSONLint** – The JSON Validator
  - https://jsonlint.com/
- **Node.js** Installation
  - https://nodejs.org/en/
  - NPM = Node Package Manager
- Install **live-server** module using NPM
  - Simple web server
  - # npm install –g live-server // install module globally
- Runs **live-server**
  - # live-server // run live-server in current directory



What is GraphQL?

GraphQL

- **Application layer query language**
  - Query the application layer or the server
- Open sourced by Facebook in 2015
- Can be used with any type of database
  - SQL, NoSQL, Hard coded data, …
- Ability to ask for exactly what you need
  - Nothing more
- Get multiple resources in a single request

Simple Query

The Query

```
{
    user(id: “100”) {
        name,
        email
    }
}
```

The Data

```
{
    “user”: {
        “id”: “100”,
        “name”: ”John Doe”,
        “email”: “john@gmail.com”
    }
}
```

Multiple Resources

The Query

```
{
    user(id: "100") {
        name,
        email,
        posts {
            title
        }
    }
}
```

The Data

```
{
    "user": {
        "id": "100",
        "name": "John Doe",
        "email": "john@gmail.com"
        "posts": [
            {"title": "Post 1", "date": "…", },
            {"title": "Post 2", "date": "…", }
        ]
    }
}
```

GraphQL Types

GraphQL APIs are organized in terms of types and fields

```
Type Query {
    user: User
}
```

```
Type User {
    name: String
    age: Int
    friends: [User]
}
```

GraphiQL Tool

- Graphical interactive GraphQL IDE
- Runs in the browser
- Syntax highlighting
- Error reporting
- Automation & Hinting

Live Demo

- http://graphql.org/swapi-graphql/

Supported Languages

- C# / .NET
- Clojure
- Elixir
- Erlang
- Go
- Java
- JavaScript / Node / Express
- PHP
- Python
- Scala
- Ruby

Who uses GraphQL?

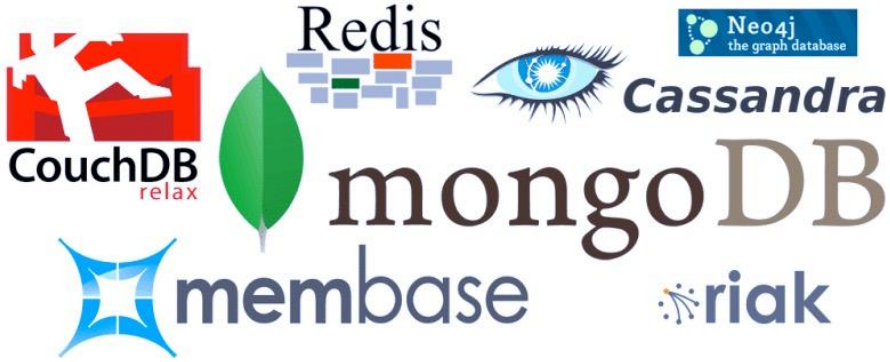
- Many huge companies use GraphQL
- Facebook mobile apps have been powered by GraphQL since 2012

Resources

- Learn GraphQL
  - http://graphql.org/learn/
- Elixer language
  - https://elixir-lang.org/
- Erlang language
  - https://www.erlang.org/
- Go language
  - https://golang.org/


mongoDB®

What is MongoDB?



- A **NoSQL** database
  - A document database
  - Use **JSON-like** syntax
  - Data or records are stored as documents
  - No schema or predefine structure of your data
- Easy to scale
- Much faster in most types of operations

Install MongoDB

- **Windows** – uses customized installation
  - https://www.mongodb.com/ (community server)
  - Install directory: "**C:\mongodb**"
  - Create subdirectories
    - "C:\mongodb\data" and "C:\mongodb\data\db"
    - "C:\mongodb\log"

Setup & Start MongoDB

- Windows – customized setup
  - Setup MongoDB using command line
    - Use command prompt as administrator
    - Navigate to "C:\mongodb\bin"
    - # mongod --directoryperdb --dbpath C:\mongodb\data\db
      --logpath C:\mongodb\log\mongo.log
      --logappend --rest –install
  - Start MongoDB server in background
    - # net start MongoDB

Mongo Shell

- Run Mongo shell command (C:\mongodb\bin)

  # mongo

- Show all data collections

  # show collections

- Create an admin user

```
# db.createUser( {
        user: “abcd”,
        pwd: “1234”,
        roles: [“readWrite”, “dbAdmin”]
});
```

Mongo Shell (cont.)

- Create a collection

```
# db.createCollection('customers');
```

- Insert documents

```
# db.customers.insert([
      { first_name: "Steven", last_name: "Smith" },
      { first_name:"Joan", last_name:"Johnson", gender="female" }
  ]);
```

- Find documents

```
# db.customers.find().pretty();
# db.customers.find({first_name:"Sharon"});
```

Mongo Shell (cont.)

- Find and counting documents

```
# db.customers.find({gender:"male"}).count();
```

- Find and sort documents

```
# db.customers.find().sort({last_name:1}).pretty();
```

- Find and process documents

```
# db.customers.find().forEach( function(doc) {
    print("Customer Name: " + doc.first_name)
});
```

Mongo Shell (cont.)

- Update documents using '**$set**' operator

```
# db.customers.update({first_name:"Steven"}, {$set:{gender:"male"}});
```

- Delete a field/property in a documents

```
# db.customers.update({first_name:"Steven"},{$unset:{age:1}});
```

- Upsert a new document if not exist

```
# db.customers.update({first_name:"Mary"},
                      {first_name:"Mary", last_name:"Samson"},
                      {upsert: true});
```

Mongo Shell (cont.)

- Delete all documents with the first_name of "Steven"

  # db.customers.**remove(**{first_name:"Steven"}**)**;

- Delete the first doc with the first_name of "Steven"

  # db.customers.**remove(**{first_name:"Steven"},**{justOne:true})**;

Q&A