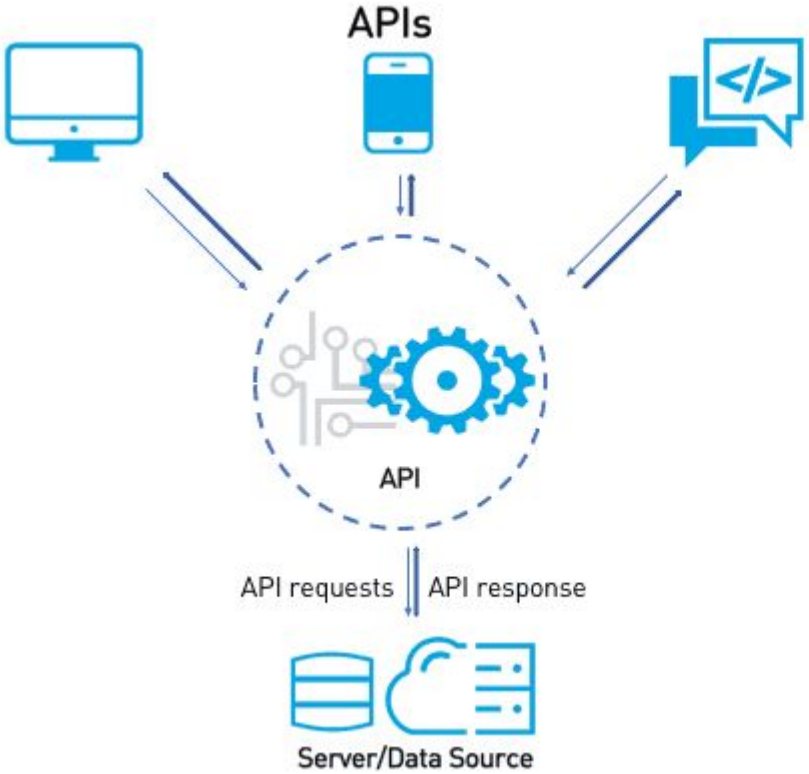
For Web Services

The Basics

Training Overview

- Web Services Overview
- What is Node.js & how to setup
- Basics of Node.js
    - **Activity -** Kahoot Quiz
- Node.js for web services with express.js
    - **Challenge** - Small Todo App
- What's Next? My recommendations for self-learning
- Closing
    - Feedback
    - Final Questions/Remarks



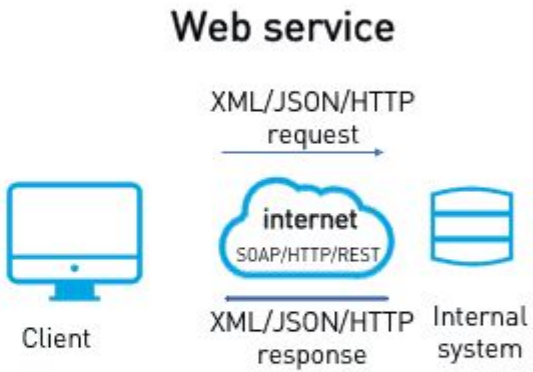


Image obtained from mulesoft.com

Web Services

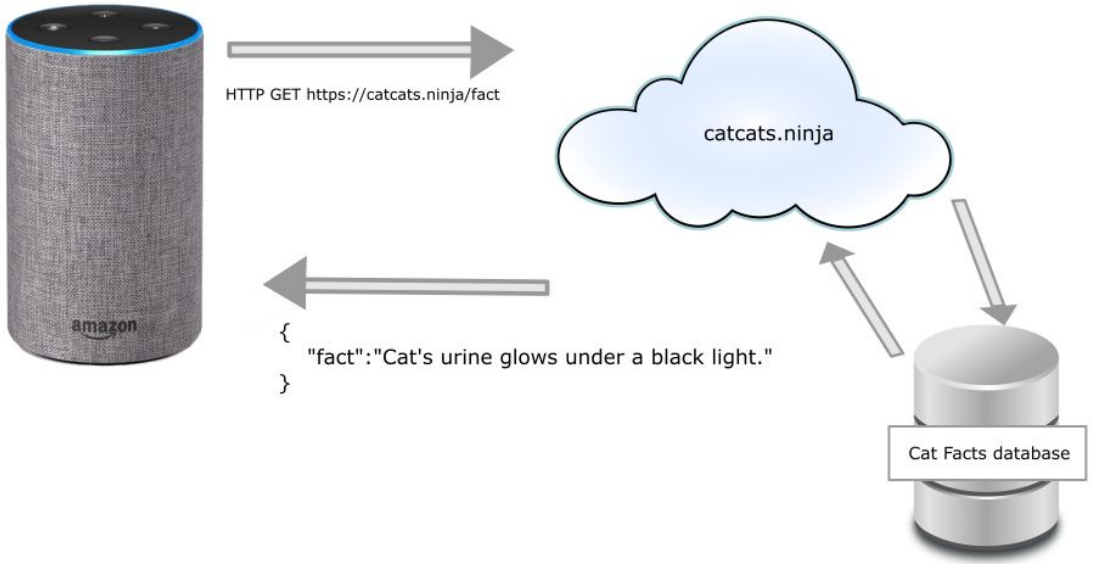
- According to IBM, “A web service is a software system that supports interoperable machine-to-machine interaction over a network... Web services fulfill a specific task or a set of tasks.”
- Basically a Web Service is a method of providing and implementing APIs over a network.
- Web Services can be used by mobile apps, Desktop Applications, other Web Services, WebSites etc.
- Difference b/w Web Service and Web Pages
    - Web Services are designed to work with other software systems, not for humans
    - Web pages are designed to have a rich user interface since they are designed for humans
    - Web pages may use multiple Web Services in order to display content on a web page

Web Services - HTTP

- HTTP
  - According to Techopedia, “HTTP is a fundamental protocol used on the Internet in order to control data transfer to and from a hosting server, in communication with a web browser.”
- Methods used in HTTP to make requests to the server
  - GET - used for reading or retrieving data or resource from the HTTP server.
  - POST - used to create a resource on the server. POST requests usually contain a request body.
  - PUT - used to update, modify or replace a resource in the server, and like POST, PUT requests may contain a request body.
  - DELETE - used to delete a resource from the server.

Web Services - basic example

"Alexa tell me a cat fact"



Web Services - For API Development

How is HTTP used to develop Web Service APIs

- URI+JSON for information transfer
  - URI is used to identify and describe the resource
  - JSON is the data format in the request or response body
- There is also XML based information transfer as API methodologies and protocols.
  - XML based Web Services: https://www.w3schools.com/xml/xml_services.asp

URI+JSON based API Example

Request

```
GET /articles?include=author HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
Content-Type: application/vnd.api+json

{
  "data": [{
    "type": "articles",
    "id": "1",
    "attributes": {
      "title": "JSON:API paints my bikeshed!",
      "body": "The shortest article. Ever.",
      "created": "2015-05-22T14:56:29.000Z",
      "updated": "2015-05-22T14:56:28.000Z"
    },
    "relationships": {
      "author": {
        "data": {"id": "42", "type": "people"}
      }
    }
  }],
  "included": [
    {
      "type": "people",
      "id": "42",
      "attributes": {
        "name": "John",
        "age": 80,
        "gender": "male"
      }
    }
  ]
}
```

Web Services - JSON basics

```
{
    "name":"Karl Amaya",
    "age": 23,
    "isStudent": true,
    "moneyOnHand":4.50,
    "fingers":["index", "thumb", "middle","pinky","rings", 5, true, { "name":"index finger" }],
    "hand":{
        "mainFinger":"index"
    }

}
```

- Valid JSON Types:
  - JSON object
  - Strings - only use double quotes
  - Numbers (integers, floats)
  - Boolean
  - Arrays - arrays can hold any of the valid JSON data types
- More on JSON: https://www.youtube.com/watch?v=iiADhChRriM

Web Services - REST as an API methodology

What is REST?

- REST is a set of guidelines that describe how communication between client and server can be constrained, REST API can be implemented in different ways.
- REST APIs are constrained by:
  - client-server architecture, whose requests and responses are managed by HTTP
  - Stateless client-server communication, no information about client is saved
  - Cacheable data to streamline communication between client and server
  - Information is transferred in a standard form
- More about REST API constraints: https://restfulapi.net/rest-architectural-constraints/

Web Services - How URIs are used in a REST API

What is a URI?

- Uniform Resource Identifier (URI) provides a simple and extensible means for identifying a resource.

Examples:

- https://example.com/users
- https://example.com/file/file.pdf
- https://example.com/users/4

URI Parameters - identify a resource or resources, often a sub resource

- https://example.com/users/{userId}
- https://example.com/students/{studentId}/parents/{parentId}

Web Services - How URIs are used in a REST API

Query Strings/Parameters - used to filter, sort or describe how the resources are returned

- Key value pairs, `key=value`
- Multiple query strings are separated by a `&`
- To begin defining query strings in a URI, you place a `?`, after which you list off your query strings
- Example: https://example.com/users?filter_active=true&sort_age=asc

Introduction to

Intro. To Node.js

What is node.js?

- Is a JavaScript runtime that uses the chrome V8 engine.
- Allows for javascript to run on your machine much like scripting languages like python.
- Created by Ryan Dahl :)
- Not too confident in your JavaScript skills?
    - “Absolute javascript for beginners” & “What even is javascript?”

Intro. To Node.js - Setup

Download Node.js from https://nodejs.org/en/

- Access Node.js from CMD or PowerShell (windows)

Download and install vscode text editor from https://code.visualstudio.com/

- Open a terminal window and access the node REPL

Install postman to test our Web Service APIs

Node.js Basics

- Package management with NPM
    - Package.json file
        - Initializing a project
            - `$ npm init`
        - dependencies
            - `$ npm install <package-name> --save `
        - Dev dependencies
            - `$ npm install <package-name> --save-dev `
        - Automation with run
            - `$ npm run <script key name>`
        - Removing a dependency or package
            - `$ npm uninstall <package-name>`
    - Global package install
        - `$ npm install --global <package-name>`
    - More on NPM: https://nodesource.com/blog/an-absolute-beginners-guide-to-using-npm/

Node.js Basics - working with packages

- Including installed packages
  - require('path to module')
- Including your own Javascript files
  - `modules.exports` for require syntax
  - Filepath - the `__dirname` global variable
- Find out about ES6 import syntax for including files and modules.
  - https://www.geeksforgeeks.org/how-to-use-an-es6-import-in-node-js/

Node.js Basics - Working with files

- Working with Files
  - The `fs` module
  - Reading from a file
    - `fs.readFileSync(filepath)`
    - `fs.existsSync(filepath)` - check if a file or file path exists
  - Writing to a file
    - `fs.writeFileSync(filepath, stringData)`
  - Working with JSON files
    - `JSON.stringify(Object)` - converts a javascript object into a JSON string
    - `JSON.parse(JSONFormattedString)` - converts a JSON string to javascript object
  - More on fs: https://nodejs.org/api/fs.html

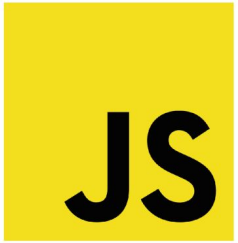
Node.js Basics - making HTTP requests

- What if our web service wants to contact other web services?
    - We can use the axios package to make HTTP requests
    - Install axios with npm: `$ npm install axios --save`
- Axios uses Promises
    - Server requests take time
    - Javascript is not patient so it won't wait for server to respond
    - Promises provides us with a way to define code that will run once the server has responded
    - More on Promises: https://youtu.be/DHvZLI7Db8E

Activity 1 - Kahoot, 15 question quiz

Web Services using

Express



Nodejs Express.js Basics - creating a server

- Create a new folder to use as a project
- Initialize your package.json
- Install express.js `$ npm install express --save`
- Use express.js to create a server

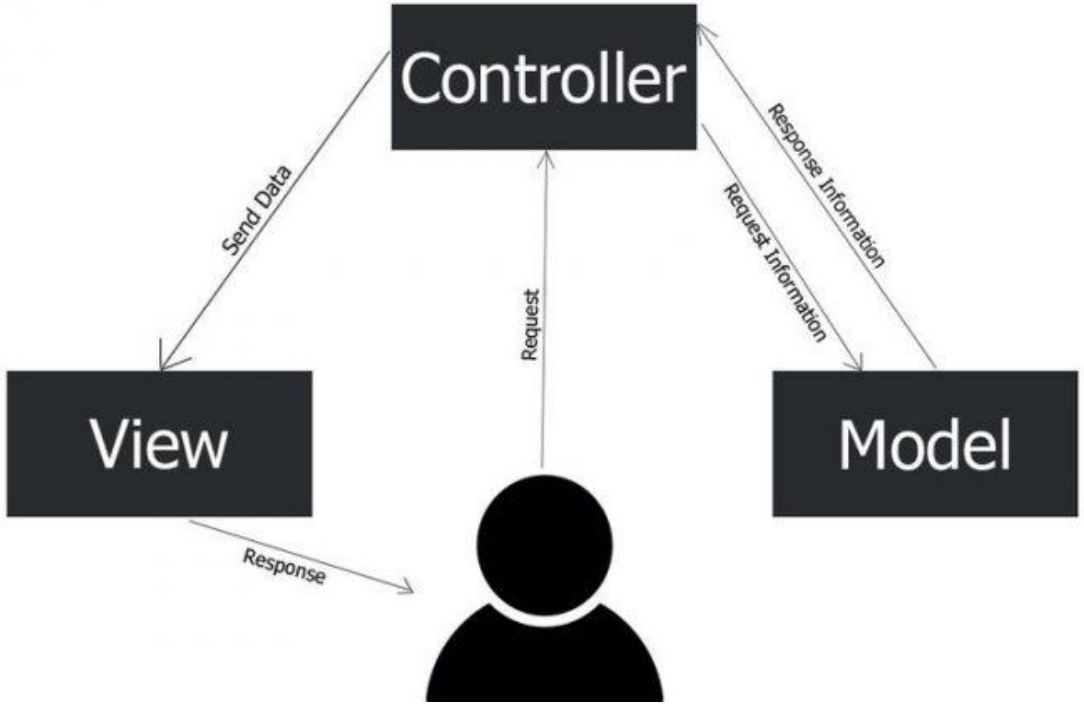
Express.js Basics - Routing

- Defining routes
  - Defining routes with URI parameters
    - Accessing our URI parameters
      - `request.params`
    - Using Query Strings
      - `request.query`
  - Using GET, POST, PUT, DELETE within express.js
    - How to access request body
      - Add `express.json()` middleware first
      - `request.body`

Express.js basics

- Defining our own middleware
  - Using `next()` method
- Planning for scalability
  - Using the Router object
  - Establishing a Model View Controller (MVC) Structure
    - Model - Database Interface
    - Controller - receive user input then tie it to business logic
      - In express this would be our route handler functions
    - View - the presentation layer, the output
      - The representation of the resource, can be a file or the data we send back to the client.

Model-View-Controller



Self-Learn Challenge- Small Todo App

- Read the README.md file inside the todoApp folder.
- Task:
  - Read the source code and try to understand it
  - finish the todo app
  - Implement an MVC structure for the todo app

What’s Next? My recommendations for self-learning

Self-learning

- Review The Todo App db.js, Model.js, and TodoModel.js for better understanding.
- Checkout all the links provided in the slides to other resources.
- Use a relational database instead of a .json file as the todo app's database
- Learn about Node.js Authentication methodologies
    - JWTs, Session-based authentication, OAuth
    - Passport.js for implementing the above methodologies
- Other web frameworks for Node.js
    - Checkout adonis.js
    - Also look into headless cms, such as strapi
- Try build your own web service using express.js
    - Something small or something big
    - For semester projects use nodejs if you can (advanced database)

Closing: Feedback

Feedback URL: https://forms.gle/4kdv47gWWXe3avyV7

Closing - Final Questions/Remarks?

Thank You and Stay Learning!

Node.js and Express.js - Full Course - YouTube

Git Repo With Code Examples

https://github.com/amayakarl/nodejsTraining

Find me on Linkedin or email me at aakarl32@gmail.com