EXPRESS.JS 101

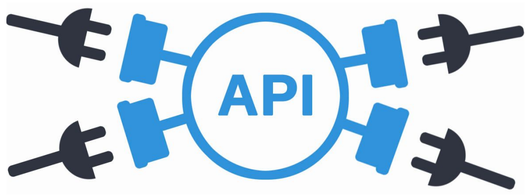
TRAJECTORY

- APIs
- HTTP
- Build a server using Express
- Scaffold out a couple of endpoints (APIs)

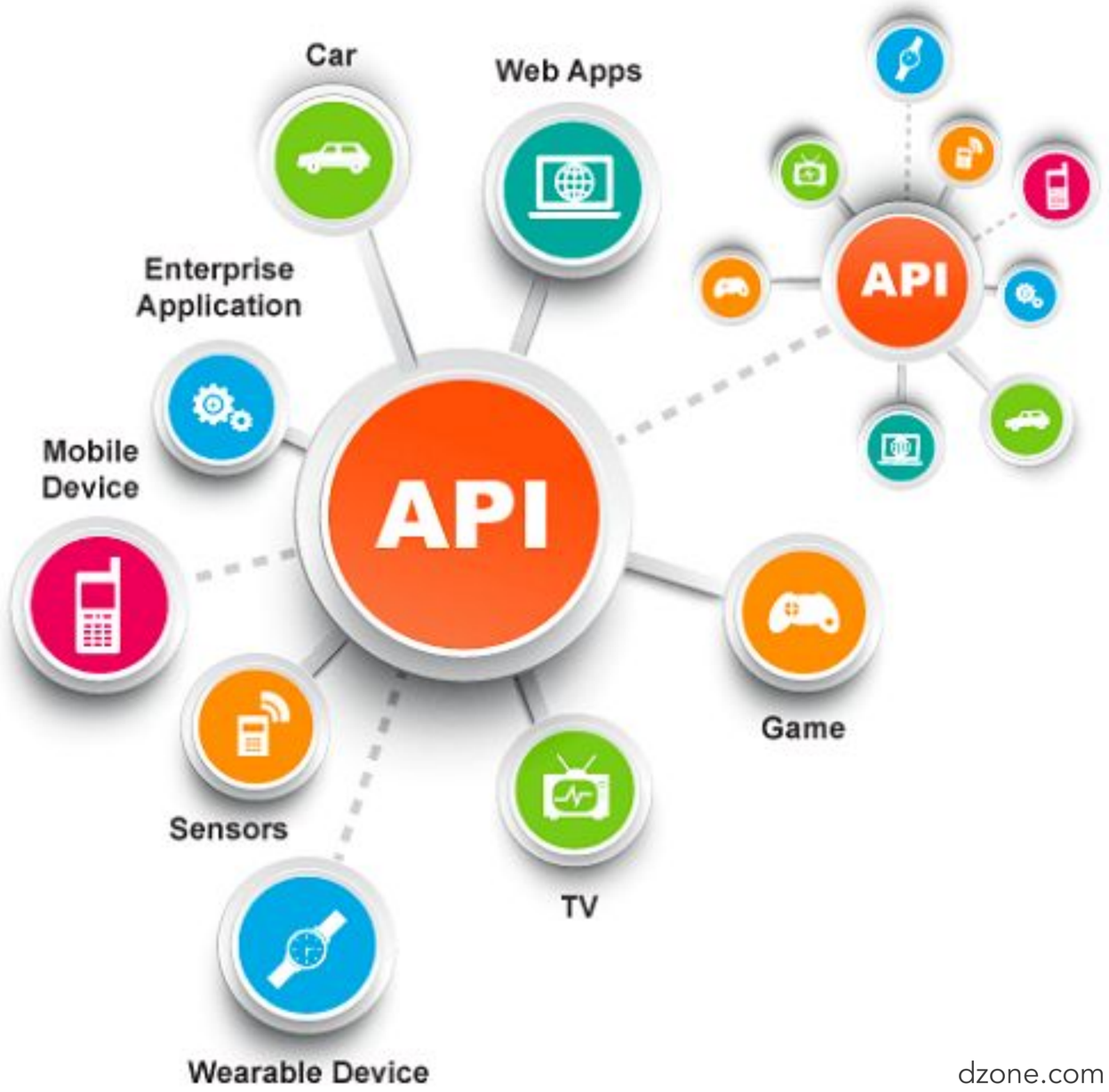
BUT FIRST...

What is  ?

- Open-source, cross-platform runtime environment that allows developers to create all kinds of *server-side* tools and applications in JavaScript
- The runtime is intended for use *outside* of a browser context
- You can use Node.js to create a simple web server using the Node HTTP package

What is an  ?

- Application programming interface
- What one application *exposes* to another (we decide this as engineers)
- A single API can be used to provide data for a variety of front-ends
  - e.g. web, iOS, and Android applications


Car
Web Apps
Enterprise Application
Mobile Device
API
API
Sensors
Game
TV
Wearable Device

dzone.com

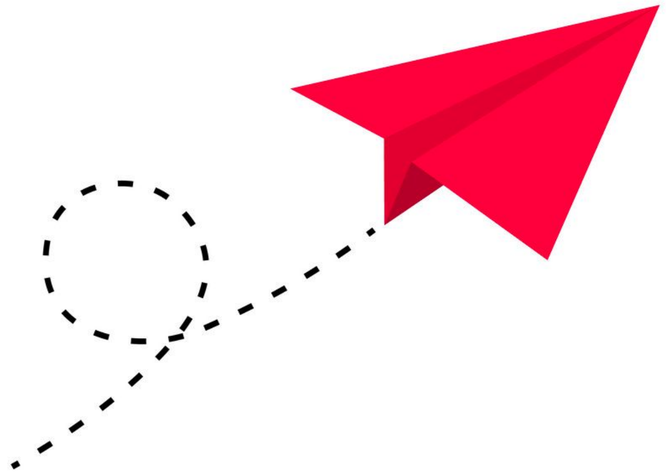
HOW DO WE ACCESS THESE APIs?

HTTP

- Hypertext Transfer Protocol (HTTP) is the *foundation* of any data exchange on the Web

- A protocol for transmitting resources (e.g. HTML, plain text, images, JSON)
  - Specifically, a client-server protocol, which means requests are initiated by the recipient, usually the Web browser
    - The messages sent by the *client* are called *requests*
    - The messages sent by the *server* as an answer are called *responses*

WHAT DO HTTP MESSAGES LOOK LIKE??



HTTP REQUEST

just a message with a certain format

```
POST /docs/1/related 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 (compatible; MSIE 6.0; Windows NT 5.1)

bookId=12345&author=Thoreau
```

(from http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html)

“Oh boy! Parsing this string of text will be lots of fun!”

– NO DEVELOPER EVER

BUILDING A SERVER WITH NODE

- Node.js provides several low-level tools for building servers
- Abstracts HTTP requests and responses into nice objects that have useful methods

```
const http = require('http')

const server = http.createServer()

server.listen(3000, 'localhost')
```

```
const http = require('http')

const server = http.createServer()

server.on('request', (req, res) => {
  res.writeHead(200)
  res.write('<h1>Welcome to my website</h1>')
  res.end()
})

server.listen(3000, 'localhost')
```

```
const http = require('http')

const server = http.createServer()

server.on('request', (req, res) => {
  if (req.method === 'GET') {
    if (req.url === '/') {
      res.writeHead(200)
      res.write(`<h1>Welcome to the main page!</h1>`)
      res.end()
    } else if (req.url === '/puppies') {
      res.writeHead(200)
      res.write(`<h1>Welcome to the puppies page!</h1>`)
      res.end()
    }
  }
})

server.listen(3000, 'localhost')
```

EXPRESS

What is express ?

- The most popular *Node* web application framework
- Provides mechanisms to:
  - Write handlers for requests with different HTTP verbs at different URL paths (routes)
  - Set common web application settings like the port to use for connecting
  - Add additional request processing “middleware” at any point within the request handling pipeline

```
const express = require('express')

const app = express()

app.listen(3000)
```

```
const express = require('express')

const app = express()

app.get('/', (req, res, next) => {
  res.send(`<h1>Welcome to the main page</h1>`)
})

app.listen(3000)
```

```
const express = require('express')

const app = express()

app.get('/', (req, res, next) => {
  res.send(`<h1>Welcome to the main page</h1>`)
})

app.get('/puppies', (req, res, next) => {
  res.send(`<h1>Welcome to the puppies page</h1>`)
})

app.listen(3000)
```

DEMO

SUMMARY

- We can use Node to easily create servers using the built-in HTTP module
- *However*, the Express library makes it even *easier* to write (and subsequently maintain) our server code
- The Express library is just a (light) abstraction over the built-in Node HTTP module, which is itself an abstraction over some code written in C++ making lower-level system calls