node

JS

Training Agenda

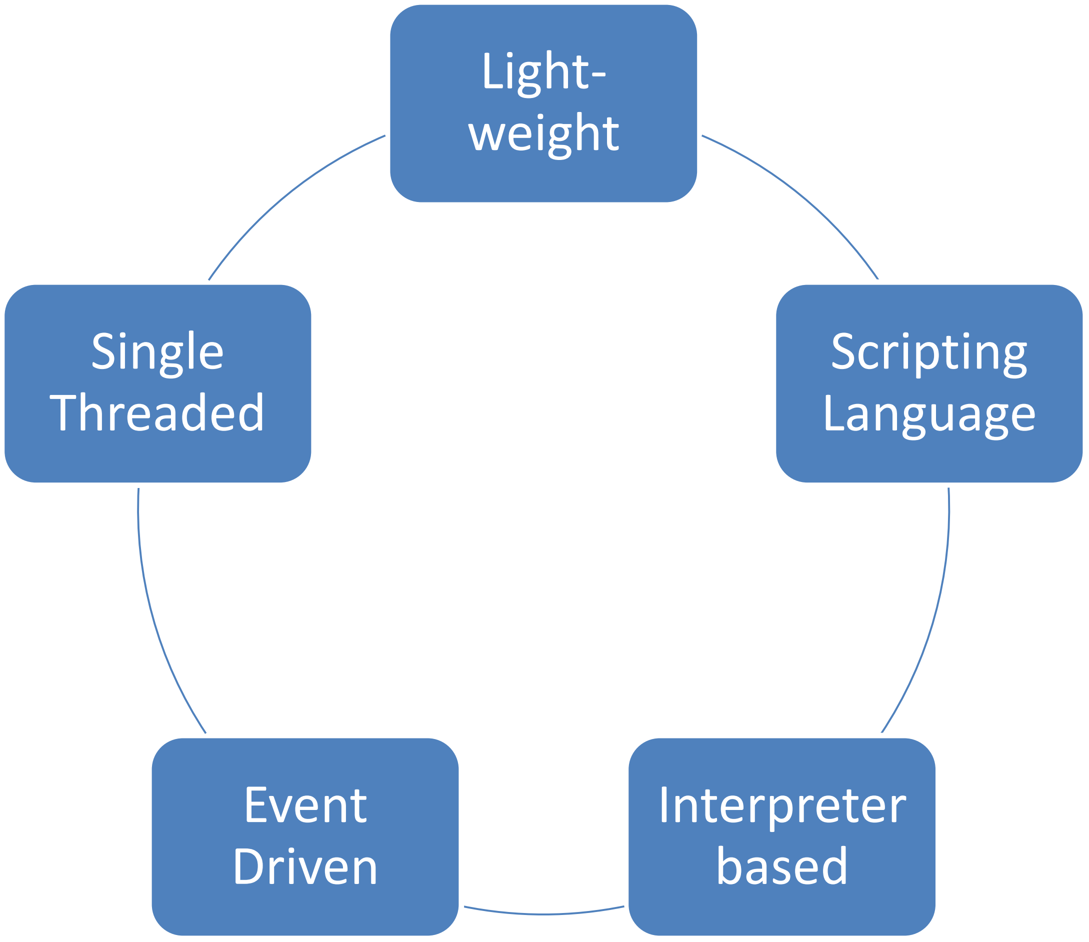
- ❑JavaScript Overview
- ❑NodeJS Overview
- ❑Modules System
- ❑File System
- ❑Buffers & Streams
- ❑Event System
- ❑Express
- ❑View Engines
- ❑NodeJS Securities

Prerequisites:

Basic knowledge of HTML, CSS & JavaScript

Interest to Learn

JavaScript Overview



JavaScript Building Blocks

Functions

- Function Expressions
- HOF
- Nested Functions
- IIFE

Objects Creation Methods

- Literal
- Constructor
- Instance

Prototyping

- Object Blueprint

Common Design Patterns

Module Design

Observer Design

Prototype Design

Singleton Design

NodeJS

“Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.”

NodeJS : Features

Extremely fast

I/O is Asynchronous and Event Driven

Single threaded

Highly Scalable

Open source

NodeJS Process Model

Node.js runs in a single process and the application code runs in a single thread.

All the user requests to web application will be handled by a single thread and all the I/O work or long running job is performed asynchronously for a particular request.

An event loop is constantly watching for the events to be raised for an asynchronous job and executing callback function when the job completes.

Internally, Node.js uses *libuv* for the event loop which in turn uses internal C++ thread pool to provide asynchronous I/O.

Module System

Use of imports/exports for importing and exporting the modules in application.

Following are a few salient points of the module system:

- Each file is its own module.
- Each file has access to the current module definition using the module variable.
- The export of the current module is determined by the module.exports variable.
- To import a module, use the globally available require function.

Require Function

The Node.js *require* function is the main way of importing a module.

The require function blocks further code execution until the module has been loaded.

Call *`require()`* based on some condition and therefore load the module on-demand

After the first time a require call is made to a particular file, the `module.exports` is cached.

Treats module as an object factory.

Module System

If something is a core module, return it.

If something is a relative path (starts with './' , '../') return that file OR folder.

If not, look for node_modules/filename or node_modules/foldername each level up until you find a file OR folder that matches something.

If it matched a file name, return it.

If it matched a folder name and it has package.json with main, return that file.

If it matched a folder name and it has an index file, return it.

Few Core Modules

path	os	fs
events	http	util

Important Globals

Console	the console plays an important part in quickly showing what is happening in your application when you need to debug it.
Timers	setTimeout only executes the callback function once after the specified duration. But setInterval calls the callback repeatedly after every passing of the specified duration.
__filename and __dirname	These variables are available in each file and give you the full path to the file and directory for the current module.
Process	Use the process object to access the command line arguments. Used to put the callback into the next cycle of the Node.js event loop.
Global	The variable global is our handle to the global namespace in Node

Node Package Manager

NPM is the eco-system to manage the project dependencies

Few NPM Commands

npm install	Install the packages/ dependencies
npm uninstall	Uninstall the packages/dependencies
npm config get/set	Gest /set the npm eco-system
npm update	Update the project dependencies
npm ls	List down the dependencies
npm search	Search the listed package on npm registry
npm init	Generates package.json file in local project directory
npm outdated	List down the outdated package

File System

Node.js includes **fs** module to access physical file system.

The **fs** module is responsible for all the asynchronous or synchronous file I/O operations.

Important Methods

Method	Description
`fs.readFile(filename, callback)`	Reads existing file.
`fs.writeFile(filename, callback)`	Writes to the file. If file exists then overwrite the content otherwise creates new file.
`fs.open(path, callback)`	Opens file for reading or writing.
`fs.rename(oldPath, newPath, callback)`	Renames an existing file.
`fs.rmdir(path, callback)`	Renames an existing directory.
`fs.mkdir(path, callback)`	Creates a new directory.
`fs.readdir(path, callback)`	Reads the content of the specified directory.
`fs.exists(path, callback)`	Determines whether the specified file exists or not.
`fs.appendFile(file, callback)`	Appends new content to the existing file.

Event System

EventEmitter is a class designed to make it easy to emit events (no surprise there) and subscribe to raised events.

Subscribing : *built-in support* for multiple subscribers is one of the advantages of using events.

Unsubscribing : EventEmitter has a removeListener function that takes an event name followed by a function object to remove from the listening queue.

EventEmitter provides a function `` `**once**` `` that calls the registered listener only once.

EventEmitter has a member function, **listeners**, that takes an event name and returns all the listeners subscribed to that event.

Creating your own EventHandler (Custom Events)

A number of classes inside core Node.js inherit from EventEmitter.

Buffers & Streams

Streams play an important role in creating performant web applications.

Improvement in user experience and better utilization of server resources is the main motivation behind steams.

All of the stream classes inherit from a base abstract Stream class which in turn inherits from EventEmitter.

All the streams support a pipe operation that can be done using the pipe member function.

Streams (Cntd…)

Readable	A readable stream is one that you can read data from but not write to.
	Process. stdin, which can be used to stream data from the standard input.
Writable	A writable stream is one that you can write to but not read from.
	Process.stdout, which can be used to stream data to the standard output.
Duplex	A duplex stream is one that you can both read from and write to.
	Network socket. You can write data to the network socket as well as read data from it.
Transform	A transform stream is a special case of a duplex stream where the output of the stream is in some way computed from the input.
	Encryption and compression streams.

Express

Web application framework for Node apps.

To create an express app, make a call require(‘express’)

Express can accept middleware using the ‘use’ function which can be registered with http.createServer.

Express Request / Response objects are derived from standard NodeJS http Request / Response

Request object can handle URL : Route Parameter & Querystrings

Express Router is used to mount middlewares and access all REST APIs

Express Middleware

Can be applied:	Application Level
	Route Level
Types of Middleware:	Built-in
	Third-party
	Error

REST Services

- REST (Representational State Transfer) is a term coined by Roy Fielding.
- REST as a general architectural style, specifying constraints on how connected components in a distributed Hypermedia system should behave.
- Web APIs that adhere to these constraints are called RESTful.
- In REST, there are two broad kinds of URLs:
  - URLs that point to collections.
  - URLs that point to an individual item in the collection.

RESTful API HTTP Method Behaviour

GET

- Get the summarized details of the members of the collection, including their unique identifiers.

POST

- Add a new item in the collection. It is common to return a unique identifier for the created resource.

PUT

- Replace the entire collection with a new collection.

DELETE

- Delete the entire collection

Express Application Routes

Express provides first-class verb + URL based routing support.

Express Routes register a middleware chain that is only called when the path + HTTP verb in the client request matches.

```
app.VERB(path, [callback...], callback)
```

all to register a middleware that is called whenever the path matches (irrespective of the HTTP verb).

View Engines

Jade	uses whitespace and indentation as a part of the syntax.
Handlebar	developers *can't write* a lot of JavaScript logic inside the templates.
EJS	Follows JavaScript-ish syntax. Embed JavaScript code in template. Commonly used.
Vash	Vash is a template view engine that uses Razor Syntax. Familiar to people who have experience in ASP.Net MVC.

NodeJS Securities

A JSON Web Token (JWT), defines an explicit, compact, and self-containing secured protocol for transmitting restricted information.

The JWT Claims Set represents a compact URL-safe JSON object, that is base64url encoded and digitally signed and/or encrypted.

jwt.sign() – signs and generate the token

jwt.verify() – verifies and provide the decoded salt

PassportJS

Flexible and modular authentication middleware for Node apps.

Comprehensive set of strategies support authentication

Authentication mechanisms, known as *strategies*, are packaged as individual modules.

Passport : Configuration Steps

- Choosing Strategy
- Initialize Strategy
- Configure the Strategy
- Verifying callbacks
- Maintain the session (serialize/deserialize)

Socket Programming

Bi-directional Full duplex communication

Continuous channel of communication

Deployment Steps - Git

- Create a new Git :
  - git init
  - git add .
  - git commit -m "initial commit"
  - git remote add origin <git>
  - git push –u origin master

Deployment Steps - Heroku

- Download heroku CLI Tool
  - heroku -v
  - heroku login

  Add RSA key
  - heroku keys:add
  - heroku create <Project_Name>

  commit all code to git

  push all code to git
  - git remote
  - git push heroku master

Clustering

Node.js is optimized for a single processor.

Cluster API utilize all the CPU cores available on a multi-core system.

Script that uses the cluster module, by default, start in *master* mode.

cluster.fork start a new child process with the *same* script as the one that is currently executing.

Ideally workers equal to the number of CPUs on your system.

Handling HTTP Requests in Workers.

Communication with the Master.

Child Process : Spawn, Fork, exec

Spawn	Fork	exec
• spawn() method spawns an external application in a new process and returns a streaming interface for I/O.	• fork() is a special case of spawn(). • Fork will have an additional communication channel built-in that allows messages to be passed back and forth between the parent and child.	• exec() method runs a command in a console and buffers the output.

Data Persistence

- Why NoSQL ?
  - Scalability
  - Ease of Development
- NoSQL servers can be placed into four broad categories:
  - Document databases (for example, MongoDB)
  - Key-value databases (for example, Redis)
  - Column-family databases (for example, Cassandra)
  - Graph databases (for example, Neo4J)

MongoDB

- A MongoDB deployment consists of multiple databases.
- Each database can contain multiple collections.
  - A *collection* is simply a name that you give to a *collection of documents*.
- Each collection can contain multiple documents.
  - A document is effectively a JSON document

```
npm install mongodb
```

Mongoose

Mongoose makes it easy to model and manage your application data.

Mongoose will serve as a replacement for the native driver, providing you with a more object-oriented interface.

Mongoose offers data sanitization, data validation & more allowing us to store data in a uniform and standardized way.

Resource creation endpoints : Creating REST

Debugging

Console Object

Debugger statement

Node-inspect

Microservices

Microservices are an architectural approach based on building an application as a collection of small services.

Smaller, lightweight pieces based on a logical construct.

Each service has its own unique and well-defined role, runs in its own process, and communicates via HTTP APIs or messaging.

Each microservice can be deployed, upgraded, scaled, and restarted independently of all the sibling services in the application.

Microservices : Pros & Cons

Pros	Each service is isolated
	Boundaries with APIs
	Developing single service is easier
	Testing and maintenance is easier
	More failure resilient
Cons	Entire application is complex
	No language level communication
	Cross service changes are hard

Testing : Mocha

“Mocha is a feature-rich JavaScript test framework running on node.js and the browser, making asynchronous testing simple and fun.”

- Mocha is best to think as a test runner (the mocha executable) as well as test API (for example, *describe, it, beforeEach* functions).

Testing : Chai

- Chai is a library that provides you with additional assertion functions.
- Good tests follow the **AAA** pattern:
  - **Arrange** all the necessary preconditions and inputs.
  - **Act** using the object / method under test.
  - **Assert** that the expected results have occurred.

References

Books :

- NodeJS by Basarat Ali Syed
- Node.JS Web Development by David Herron

Web :

- https://mongodb.github.io/node-mongodb-native/
- https://mongoosejs.com/docs
- Javascript.info
- https://nodejs.org
- http://expressjs.com/
- https://mochajs.org/api/mocha
- https://www.chaijs.com/api/
- npmjs.com
- https://stackoverflow.com
- http://www.passportjs.org/
- https://www.dofactory.com/javascript/design-patterns