ITSE412– Week 4

Node.JS – NPM

Node.js - NPM

- NPM: Node Package Manger provides two main functionalities:
  1. Online repositories for Node.js packages/modules.
  2. Command line utility to install packages and do version management and dependency management of Node.js applications.

  NOTE: NPM comes bundled with Node.js.

  To verify the NPM version:

  C:\Nodejs_WorkSpace> npm --version (or npm –v)

NPM Global vs Local Installation

- By default, npm installs any dependency in the **local mode**.
  - Local mode refers to the package installation in node_modules directory in the same folder where Node application is present.
  - Locally deployed packages are accessible via **require**.
- Globally installed packages are stored in /npm folder under the Node folder. Such packages can be used in CLI (Command Line Interface) function of any Node.js

Global or Local?

In general, the rule of thumb is:

- If you're installing something that you want to use *in* your program, using require('module_name'), then install it locally.
- If you're installing something that you want to use in the *shell*, on the command line, install it globally, so that its binaries end up in your PATH environment variable.

Install/uninstall a module

- Installation of any module is as simple as typing the following command:

  c:\nodejs_workspace> npm install module_name

- Now you can use it in your js file as following:

  var module_name= require(module_name');

- Use the following command to uninstall a module:

  c:\nodejs_workspace> npm uninstall module_name

- Note: to verify use the following command:

  c:\nodejs_workspace> npm ls

Installing Express package (local)

- Express.js, is a very popular web framework
- To install it locally in your application:

c:\nodejs_workspace\myapp> npm install express

- When npm completes the installation, express will be in the folder c:\nodejs_workspace\myapp\node_modules
- Or type the following command:

c:\nodejs_workspace\myapp> npm ls

Installing express (global)

- To install express as global package use –g

  c:\nodejs_workspace\myapp> npm install express –g

- When npm completes the installation, express will installed under the directory <user-directory>/npm/node_modules.
- Type the following command to verfy it:

  c:\nodejs_workspace> npm ls -g

Update/search module

- Use the following command to update a module in the same directory as your **package.json** file:

  c:\nodejs_workspace\myapp> npm update

- To test the update, run the command:

  c:\nodejs_workspace\myapp> npm **outdated**

- NOTE: To update all global packages, you can use c:\>npm update -g.
- Use the following command to search a module:

  c:\nodejs_workspace> npm search express

How Node Application Work?

```
var fs = require("fs");
fs.readFile('./test.txt', (err, data) => {
        if (err) { console.error(err);
                return;
        }
        console.log(data.toString());
});
console.log("Program Ended");
```

- In Node Application, any asynchronous function takes a callback as a last parameter and the callback function accepts error as a first parameter.
- Here fs.readFile is a async function. If an error occur during file reading, then err object will contain the corresponding error else data will contain the contents of the file. readFile passes err and data to callback function after file read operation is complete.

Event Loop Overview

- Node.js is a single threaded application but it supports concurrency via concept of **event and callbacks**.
- As every API of Node.js are asynchronous and being a single thread. It uses asynchronous function calls to maintain the concurrency.
- Node uses observer pattern (waiting for event).
- Node thread keeps an event loop and whenever any task get completed, it fires the corresponding event which signals the even listener function to get executed.

Event Driven programming

- Node.js uses Events heavily which makes it pretty fast compared to other similar technologies.
- As soon as Node server starts, it simply initiates its variables and then simply waits for an event to occur.
- The functions which listens to events acts as an observer. Whenever an event got fired, its listener function starts executing.

EventEmitter

- EventEmitter class lies in **events** module. It is accessibly via following syntax:

```
// import events module
var events = require ('events');
// create an eventEmitter object
var eventEmitter = new events.EventEmitter();
```

- When an EventEmitter instance faces any error, it emits an ‘error’ event. When new listener is added, ‘newListener’ event is fired and when a listener is removed, ‘removeListener’ event is fired.

EventEmitter

- EventEmitter provides multiple properties like **on** and **emit**.
- **on()** method is used to bind a function with the event and **emit()** is used to fire an event.

EventEmitter sample

```
var events = require('events');
var eventEmitter = new events.EventEmitter();
var connected = function connected() {
        console.log('connection successful.');

        eventEmitter.emit('data_receive');
}
eventEmitter.on('connection', connected);
eventEmitter.on('data_receive', () => {
        console.log('data received successfully.');
});
eventEmitter.emit('connection');
console.log('Program Ended.');
//  ===== output ======== //
connection successful.
data_received successfully.
Program Ended.
```

EventEmitter Methods

Method	Description
addListener(event, listener)	Adds a listener to the end of the listeners array for the specified event. Returns emitter, so calls can be chained.
on(event, listener)	Adds a listener to the end of the listeners array for the specified event. Returns emitter, so calls can be chained.
once(event, listener)	Adds a one time listener for the event. This listener is invoked only the next time the event is fired, after which it is removed. Returns emitter, so calls can be chained.
removeListener(event, listener)	It will remove, at most, one instance of a listener from the listener array. Returns emitter, so calls can be chained.

EventEmitter Methods (continue…)

Method	Description
setMaxListener(n)	By default EventEmitter will print a warning if more than 10 listeners are added for a particular event. This is a useful default which helps finding memory leaks. Obviously not all Emitters should be limited to 10. This function allows that to be increased . Set to zero for unlimited.
listeners(event)	Returns an array of listeners for the specified event.
emit(eventName[, arg1][, arg2][,…])	Synchronously calls each of the listeners registered for the event named eventName, in the order they were registered, passing the supplied arguments to each. Returns true if event had listeners, false otherwise.