What’s node - single threaded event driven non blocking IO

Comes with npm

*Require*s modules, core modules or third party or can create custom one

Third party modules live in npm world

Npm modules are well tested and can be assumed to be event driven

Npm projects generally have app.js entry point.

To Run: cd <app directory>

Then: node app.js <optional user arguments>

Npm init : adds package.json file to the project having build dependencies and stuff

***Npm install <modulename> --save***

Installs module in the project folder and adds it’s entry to the package json file

After installation just require and use the module functionalities

Import module name stays same as mentioned in the package json file.

Os, fs, etc. are installed project wise

Nodemon: Re-run projects in development whenever changes are made to them

Nodemon installed with flag -g instead of --save to install it globally in the node installation directory

To run in RE-RUN mode: cd <app directory>

Then: nodemon app.js <optional user arguments>

Nodemon app.js **-e js,hbs** (listens for changes in only these file extensions)

Process: has all info of current running node process

Process.argv : user cmd arguments vector

Yargs: intelligent cmd argument reader module

Eg : node app.js <arg1> --<key1>=<value1>

Reads arguments separately and maintains Json type structure for key value pairs.

Process.argv is replaced by yargs.argv

//Yargs handling , arrow functions and refactoring with try catch blocks with file handling

ES 6 template strings `string ${<jsvariables>}`

DRY principle of refactoring : don’t repeat yourself (copy pasting)

**DEBUG command line:**

*node inspect <jsfile>*

**list**(<number of lines before and aftr the current line to be printed>)

**n**: go to next line

**c**: continue debug run

**repl**: current app context for variable debug and modification

**Ctrl+c**: to move back to ebug from repl mode

Use “debugger;” statement to add a breakpoint int the node script.

Nodemon and Debug combinations are handy while development.

**Chrome Dev Tool Debugging:**

Node --inspect-brk <jsfile>

Go to chrome://inspect and open your node app

When used with nodemon, the chrome reconnects when a relaunch is performed for a change.

Advanced Yargs:

.command and .help to validate argument and provide help for the same when a call is made to this api

*//const argv = yargs.argv;*

*const argv = yargs*

*.command('add','Add a new note to NoteList',{*

*title:validations.titleOptions,*

*body:validations.bodyOptions*

*})*

*.help().argv;*

*const titleOptions = {*

*describe: 'note title',*

*demand: true,*

*alias: 't'*

*}*

**Async Non-blocking Weather application with google and other third party weather apis:**

Every method gets added tostack and all its statements are oneby one added and executed.

A statement can

perform an operation: a+b,

add a method call to stack: foo(),

remove a method call to stack:return;,

or add a callback call: setTimeout(foo(),100);

Whenever an event for callback completes, it does not call its callback right away. Instead it adds the callback call to queue which waits for the call stack to become empty and then starts executing.

Note: this is why setTimout(f(),0); executes after the last statement execution.

**Additional Notes:**

Pretty printing of JSON and other objects in code comments weather app.

//console.log(response) is of little understanding and use when printing nested objects

//Pretty printing

//<object to stringify>, whitelist optional parameter, indentation spacing size

Request & Response:

Request response body and headers.

Response code, Request, Body and Headers make up a complete RESPONSE.

Yargs: URL encoding decoding and mandatory user input handling

Handling Responses:

Errors:

Machine Error on Client side due to connectivity issues. This comes back in error argument of response callback.

Server errors sent by server for different types of issues sent back in response object with status and status codes.

Try different types of inputs and request data to create handling scenaris while working on consumption of a new api.

Refactoring is needed and is useful in most of the cases when handling apis.

Promises:

One time callbacks(max one arg), reduced risk of callback being called again or both success and error callbacks being called for same async call.

chained promise calls

\*error handler of one leads to calling of success of the other as it thinks error handling has settled everything

\*hence we replace the duplicate error handlers with single catch call in the end.

AXIOS:

Library working with async requests and promises.

Chaining instead of nesting of promises and hence a bit of easier handling of coding and error handling for any server/connection error or any error thrown by us.

**Server set up and Deployment**

**Node-web-server project**

Express JS: For reference refer to expressjs.com

Var app = express()

Now this app represents our server instance

app.listen(port,()=>{//optional second parameter function})

And many other configurations

app.get(‘/url’,()=>{})

listener /handler for the url hit

Serving  static content using middle ware

app.use(express.static(dir)) address of public directory using \_\_dirname global variable

Injecting values for Dynamic templates(web page skeletons).

Viewing engine for express => handlebar(and many more like ejs or pug) handlebarsjs.com

Actual package **hbs** which is a wrapper plugin for handlebars to be used in express

response.render(‘hbs template name in the views directory’, object(optional))

This object has the info to be injected to the template

Route handling: /, /about etc. along with a listener method.This can be further refactor to meet purposes.

Setting up a static server to serve resources.

Templates: HBS Handlebars plugin

Populating template with reusable templates: partials

Sending data to partials: template helper

This is registering functions just like a var foo = function(){}

hbs.registerHelper(key, ()=>{ return whatever})

Now use this as {{key}} anywhere in your template to get value form this function and now you **dont** need to end this value with response.render(‘hbs template’, {**HERE**})

Helper call with argument {{helpername **argumentvariableorconstant**}}

**MIDDLEWARE:**

**app.use()**  to register middlewares and they are executed in order they are declared.

We used express middleware for adding additional functionalities to our server.

These middlewares can be a third party ones or we can create one ourselves.

*//custom middleware*

*app.use((request, response, next)=>{*

*//logger as a middle ware to log the timing of server requests*

*var currentTime = new Date().toString()*

*console.log(`${currentTime}:${request.method} ${request.url}`);*

*//sends the control to next handlers and continues the process of setting up and starting the server*

*next();*

*});*

GIT: git-scm.com

Use git bash

Make key in ~/.ssh i.e., the users/rishabh directory

ssh-keygen -t rsa -b 4096 -C ['risharma04@gmail.com](mailto:'risharma04@gmail.com)'

*$ eval "$(ssh-agent -s)"*

Agent pid 5880

*$ ssh-add ~/.ssh/id\_rsa*

Identity added: /c/Users/Rishabh/.ssh/id\_rsa (/c/Users/Rishabh/.ssh/id\_rsa)

Now login to github account and in settings add a new ssh key by copuing the contents of your <>.pub file

Testing ssh setup:

*$ ssh -T git@github.com*

The authenticity of host 'github.com (192.30.253.112)' can't be established.

RSA key fingerprint is SHA256:nThbg6kXUpJWGl7E1IGOCspRomTxdCARLviKw6E5SY8.

Are you sure you want to continue connecting (yes/no)? yes

Warning: Permanently added 'github.com,192.30.253.112' (RSA) to the list of known hosts.

Hi risharma1! You've successfully authenticated, but GitHub does not provide shell access.

After this create a new repo on github and use the commands to import from existing repository.

*$ git remote add origin https://github.com/risharma1/node-web-server.git*

To specify the remote URLs you want to sync your code with For eg. syncing code to software versioning and deployment remotes.

*$git push -u origin master*

This command may ask you for credentials if login fails.

Sample response:

Username for 'https://github.com': risharma1

Counting objects: 14, done.

Delta compression using up to 4 threads.

Compressing objects: 100% (12/12), done.

Writing objects: 100% (14/14), 2.20 KiB | 752.00 KiB/s, done.

Total 14 (delta 2), reused 0 (delta 0)

remote: Resolving deltas: 100% (2/2), done.

To https://github.com/risharma1/node-web-server.git

\* [new branch]      master -> master

Branch 'master' set up to track remote branch 'master' from 'origin'.

**DEPLOYMENT:**

Heroku cloud deployment solution.

Create an account and install command line tools(toolbelt.heroku.com) to open deploy and sync apps seamlessly.

Cmd:

*heroku --help*

*heroku login*

Id

Pwd

*heroku keys:add*

*heroku keys*

*heroku keys:rmeove <email for key>*

Test:

ssh -v [git@heroku.com](mailto:git@heroku.com)

When deploying apps, the port on which the app listening is gonna bind to, is not fixed and changes dynamically. This info is exported to system environment variable PORT. We will use this one to start our app by fetching the value for the same in **server.js**.

Add “start” script as “node server.js” to package.json scripts section for heroku to run as start script when we deploy.

Also we can run via ***npm start*** now.

Heroku create(in cmd)

Git push heroku (in bash)

My case o/p had this url:  <https://fathomless-lowlands-37539.herokuapp.com/>

Can use this or heroku open to open app.