**What is JavaScript?**

JavaScript is a programming language which run on browser, It (JavaScript engine in browser) interprets the code into machine code.

Most popular JavaScript engine is V8 which was developed by Google.

**Why JavaScript??**

1. JavaScript has certain characteristics that make it very different than other dynamic languages, namely that it has no concept of threads. It’s model of concurrency is completely based around events.

**Question**

Why concurrency through events is better than threads?

**Scenario**

Below tasks are assigned to a thread. A thread can only perform one task at a time

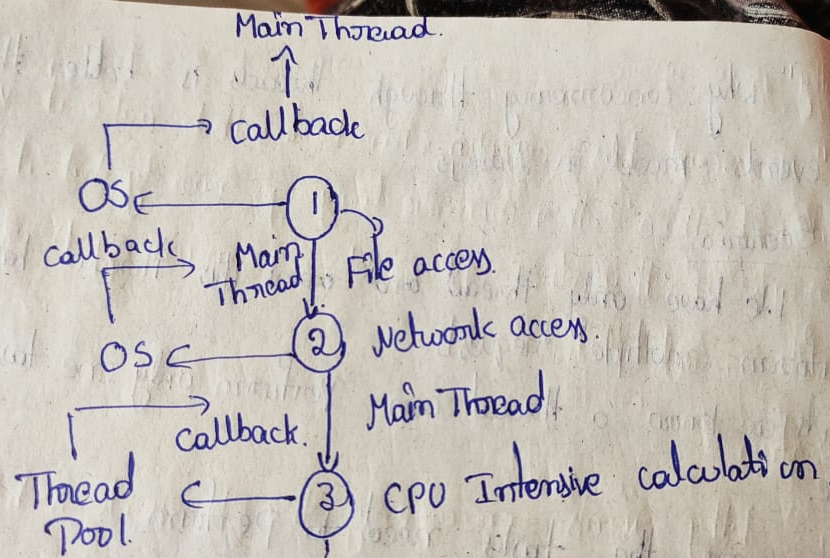
1. Read a file – Task1
2. Request another HTTP server/service
3. Perform some CPU intensive calculations

Thread-1 is assigned with the task-1 even though task-1 is done by OS(I/O Operation). We can’t make use of the thread for other tasks until it completes the current work. So, It is blocking other tasks to performs increases the response time of request.

**Problem** - One task is blocked by other task even though thread is idle

**Solution**

If we have to convert a thread in such a that it can make non-blocking calls. Below is the architecture of it



In the above example Main Thread achieve the non-blocking calls by delegating the tasks to OS and Thread Pool and then operate on the results received in the call backs.

1. The tasks which cannot be performed through OS are done through thread pool.
2. All the tasks are performed asynchronously thus non-blocking is achieved.

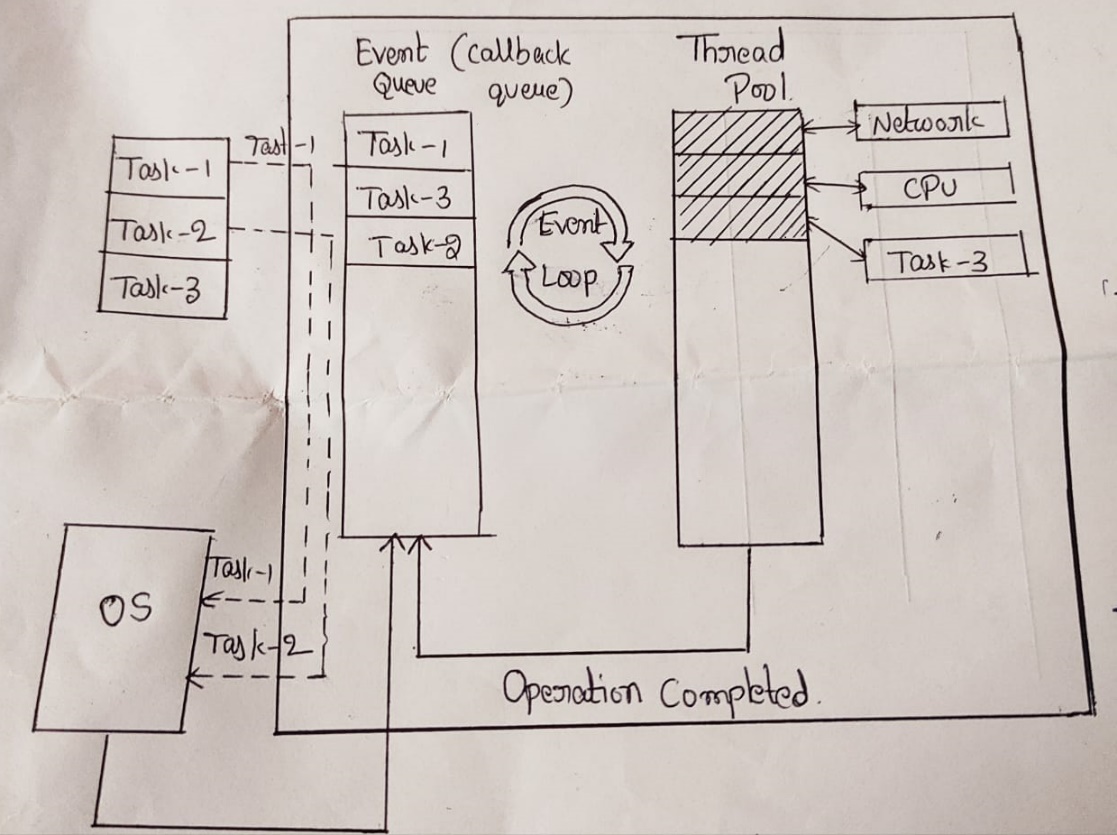
**What is Node.js**

Node.js is runtime environment to run JavaScript on stand-alone machines(local machines) instead of browser.

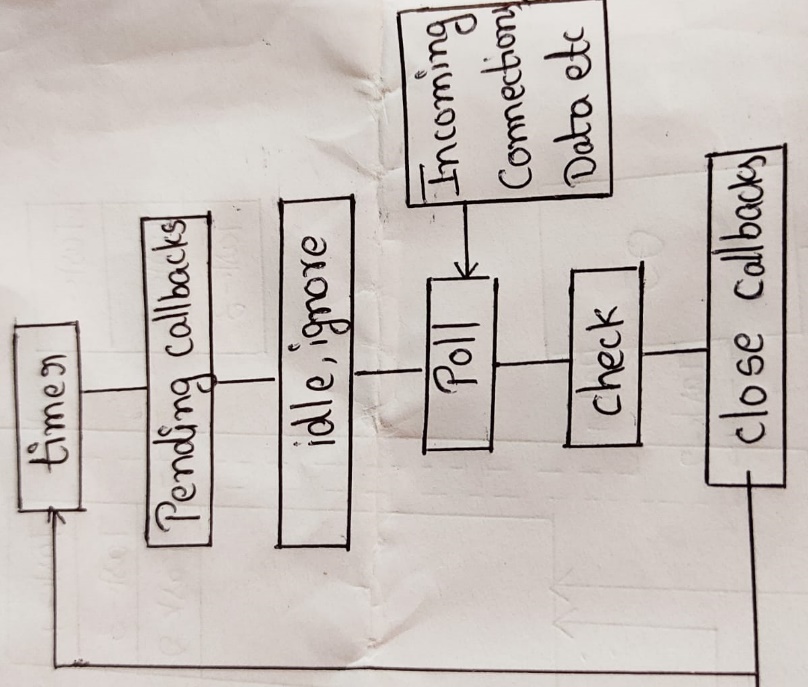
**Why Node.js**

1. It builds scalable network applications using JavaScript on server side
2. It is very fast because it is completely written in C

The above-mentioned non-blocking designee is implemented in node.js through event loop



**Phases of event loop**



**Timers –** Call-backs scheduled by setTimeout() or setIntervel() are executed in this phase

**Pending call backs –** I/Ocallbacks deferred to next loop iteration are executed here

**Idle, ignore –** It is used internally

**Check** – It invokes setIntermediate() callbacks

**Close call backs –** It handles some close call backs

ex- Socket

In the above example node.js thread assigns two I/O tasks to OS and CPU bound task to thread pool to perform tasks asynchronously. After completion of each task call back event of corresponding task is added to event queue. Even loop starts its execution only if call stack is empty.

**Features of event loop**

1. Event loop is endless loop, which waits for tasks, executes them and then sleeps until it receives more tasks.
2. Event loop allows us to use promises and call backs
3. Event loop executes the tasks starting from the oldest first.

**Call back function**

Call back function is used to deal with multiple requests made to the server.

**Example**

**Customer-1**

**Server Chef**

**Customer-2**

Suppose customer-1 order or made a request to server. Server inform to chef and dish will take 10 minutes, So server informed to chef that meanwhile will take up other requests, please call me back once dish is ready. Same concept is applied in node.js call backs as well.

**Express js**

Express.js is framework to develop web applications

**Framework** – It provides essential supporting structure of a building or object to the context.

Difference between Framework, Library, Programming Language and Package.

1. **Module** – Set of methods, functions ready to be used somewhere else. It is a part of a software
2. **Package** – Package is collection of modules with similar functionality
3. **Library** – It is collection of packages
4. **Framework** – Collection of library

**Difference between library and framework**

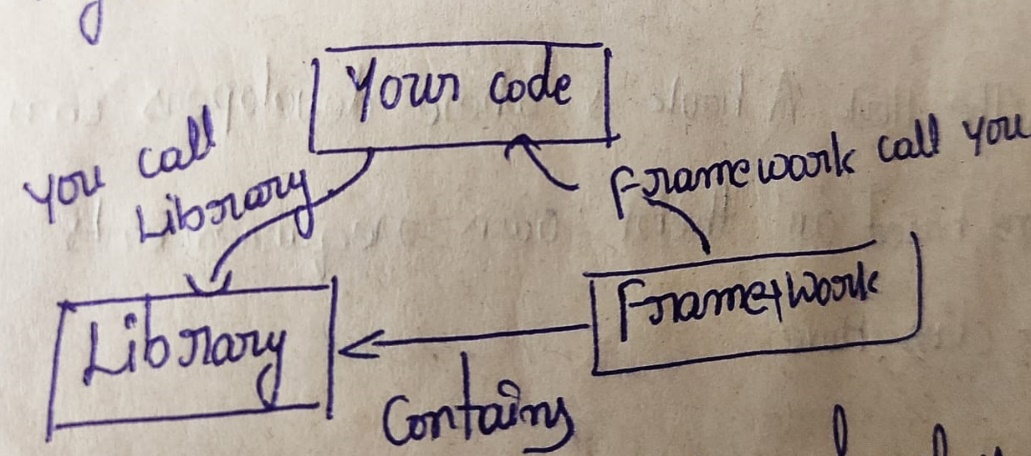
1. Major difference is “Inversion Control”

**Library**

1. A library is just a collection of class definitions.
2. It is designed for code reusability. It will get the code that has already been written by other developers
3. It is already developed. People who need this just call this library and get the result without any code implementation

**Framework**

1. All the control flow is already there
2. There is a bunch of predefined white spots that you should fill out with our code
3. Developers no need to worry about if a design is good or not but just about implementation of domain specific functions.



**Example**

1. **Main.java –** Entry point to the frame work. It cannot be changed.

Public class Main

{

Public static void main(String[] args)

{

Human h = new Human(new Walk());

h.doMove();

}

}

1. **Move.java** – It is the hook. A hook is where developers can defined/extend functions based on their own requirements.

Public abstract class Move

{

Public abstract void action();

}

1. **Human.java –** Itis the template, which reflects the idea of how the framework works.

Public class Human

{

Private Move move;

Public Human(Move m)

{

This.move = m;

}

Public void doMove()

{

This.move.action();

}

}

In this simple framework, action() method is the only thing developers are able to change.

1. **Client Code**

Public class Walk extends Move

{

@override

Public void action()

{

System.out.println(“5 miles per hour”);

}

}

**Standard Package –** It is usuallydeveloped for a particular task. A good example is the nodemon package for node.js and express

1. nodemon package automatically restarts the server after saving changes, So you don’t have to go through the process over again

**Library –** Ex – jQuery – It is more or less a group of js functions chunked together in one file to make it faster when coding in javascript

**Features of Express js**

1. Web server feature
2. Easy development
3. Middleware supportive
4. Robust Routing

**What could you build with express and node.js**

1. WebSocket Server
2. Flat file upload client
3. Any real time data apps

**Project setup using node.js and express.js**

1. Create a new folder and initialize npm using below command which will create a package.json file

npm init

1. Install required modules specific to your project(internally) using below command

npm install <module-name> save

if you want install modules gloabally use below command

npm install -g <module-name>

1. Install dev dependencies(which are required only in development environment).

Ex - nodemon

npm install nodemon --save-dev

1. Set up your start and dev scripts to run your apps easily
2. Add below key value pairs under scripts section in package.json

"scripts":

{

"start": "node ./src/index.js",

"dev": "nodemon ./src/index.js -e js,hbs"

}

b. Now use below commands to run your app

i. npm start

ii. npm run dev

**What is package,json file??**

1. All npm packages contain a file, usually in the project root, called package.json, which file holds various metadata relevant to the project
2. It contains all the information related to project as well as dependencies
3. It also contains other meta data such as
4. Project Description
5. Version of the project in a particular distribution
6. License information
7. Configuration data
8. All the above mentioned information is vital to both npm and to the end users of the project or package

**NPM**

1. npm is package manger for the node javascript platform
2. It puts modules in place so that nodes can find them
3. It is used to publish, discover, install and develop node programs
4. npm install all the packages from below registry

<https://registry.npmjs.org>

command – npm config get registry

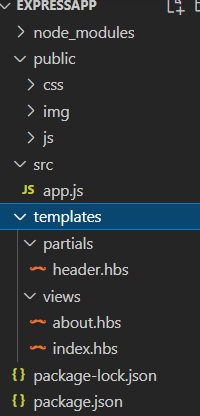
**Template Engine**

1. A template engine enables you to use static template files in your application.
2. At runtime, the template engine replaces variables in a template file with actual values, and transforms the template into an HTML file sent to the client. This approach makes it easier to design an HTML page

Package Manager for different technologies as follows

1. Python – Pypi
2. PHP – PEAR
3. RUBY – gems

**Folder structure for creating simple express app**



**public –** All the static files placed here (img,css,js). We must set this path serving up static files using below command in app.js or index.js file

const publicDirectoryPath = path.join(\_\_dirname,'../public')

app.use(express.static(publicDirectoryPath))

**src –** It is root folder for any applications. It contains root file app.js or index.js and utils folder, which contains other server side js files.

**templates** – All templates required for the app are placed in this folder It has mainly two sub-folders

1. **partials** - Partials are basically just views that are designed to be used from within other views. Below is syntax for reusing

{{>partial }}

1. **views –** It contains all endpoint views of the application

Below code must be added in app.js or index.js file to work partials and views

const viewsDirectoryPath = path.join(\_\_dirname, '../templates/views')

const partialssDirectoryPath = path.join(\_\_dirname, '../templates/partials')

//Setting View Engine

app.set('views', viewsDirectoryPath);

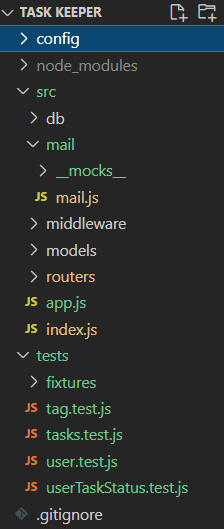
app.set('view engine', 'hbs')

hbs.registerPartials(partialssDirectoryPath)

**Reference** - <https://github.com/SRUJANA-PENUGONDA13/ExpressApp>

**Full Stack node.js and express.js project**

Below is the folder structure for the



**config –** This folder contains configuration files for dev, test and production. We must add below lines in package.json file to access your config files while running your application

"dev": "env-cmd -f config/dev.env nodemon src/index.js",

    "test": "env-cmd -f config/test.env jest --watch --runInBand"

**src** – It contains root file to start the application and below sub folders

**db –** This folder containsall the files related to establish database connection

**middleware** – All the middleware files are placed in this folder

**Express middleware** are functions that execute during the lifecycle of a request to the **Express** server

**Simple Middleware example:**

1. var app = express();
2. app.use(function(req,res,next)){
3. console.log("Request URL - "req.url);
4. next();
5. }

The above code would be executed for each request that comes in and would log the request url, the *next()*method essentially allows the program to continue. If the *next()* function is not invoked, the program would not proceed further and would halt at the execution of the middleware.

**models –** All the schemas of database are stored here

**routers** – All the files related to routers(end points) are stored here

**tests** – It contains all the test scripts to test the application in dev environment

**fixtures** – It contains all the files used to save data before running tests and delete data after tests are completed.

**\_\_mocks\_\_** - It contains all the manual mock scripts

Manual mocks are used to stub out functionality with mock data. For example, instead of accessing a remote resource like a website or a database, you might want to create a manual mock that allows you to use fake data. This ensures your tests will be fast and not flaky.

**JEST** - **Jest** is a JavaScript testing framework designed to ensure correctness of any JavaScript codebase

**MongoDb**

1. Mongo Db is NoSQL(Not Only SQL) database
2. Terminology comparison between NoSQL and SQL database
3. Row/Record - Document
4. Table - Collection
5. Column – Field
6. After installing mongodb copy the folder to your users folder and rename it if required (mongodb)

and also create new folder for data (mangodb-data)

1. Run below command to run your server initially

/c/Users/mongodb/bin/mongod.exe --dbpath=/c/Users/mongodb-data

1. We must restart the mangodb once we make any changes to schema or database

Note – mongodb is freely can be hosted on mongodb-atlas cloud

Reference - <https://github.com/SRUJANA-PENUGONDA13/Task-Manager-API>

**Web Sockets**

Socket.IO is a JavaScript library for real time web applications. It enables real time, bi-directional communication between web clients and servers.

It has two parts: a client-side library that runs in the browser, and a server-side library for node.js.

Both components have a nearly identical API.

1. Wen Sockets allow us to provide full-duplex communication i.e. bidirectional

- Server can make a request and get the data from client and vice versa

Where as in http protocol client made a request and server process it sends the data only unidirectional communication is possible

2. It provides persistent connection between client and server

3. Install socket.io

npm i socket.io

4. Create server using http module, add app to the server

const server = http.createServer(app)

5. Add server to the socket

const io = socketio(server)

6. socket.io/socket.io.js is a client side library for socket to enable bidirectional communication. Add it in your html

7. We should call client side library to register to server through socket. Create a js file and call io().

8. Main concept is server emits events client receives it and client emits event and server receives

9. Below code snippet sends the message to all the users in the room

io.to(user.room).emit('update-status', user,"Left")

10. Below code snippet sends the message to all the users except the current user

socket.broadcast.to(user.room).emit('add-player', data)

11. Below code snippet sends the message to only the current user

socket.emit('add-player', data)

**GIT Commands**

1. Installing Git
2. Tell Git who you are.
3. git config --global user.name "YOUR\_USERNAME"
4. git config --global user.email "im\_satoshi@musk.com"
5. git config --global --list # To check the info you just provided
6. Generate/check your machine for existing SSH keys.
7. Enter ls -al ~/.ssh to see if existing SSH keys are present:
8. Paste the text below, substituting in your GitHub email address.

ssh-keygen -t ed25519 -C "your\_email@example.com"

1. When you're prompted to "Enter a file in which to save the key," press Enter. This accepts the default file location.
2. Enter a file in which to save the key (/c/Users/you/.ssh/id\_ed25519):[Press enter]
3. At the prompt, type a secure passphrase. For more information, see "Working with SSH key passphrases".
4. Enter passphrase (empty for no passphrase): [Type a passphrase]
5. Enter same passphrase again: [Type passphrase again]
6. Adding your SSH key to the ssh-agent
7. Ensure the ssh-agent is running. You can use the "Auto-launching the ssh-agent" instructions in "Working with SSH key passphrases", or start it manually:
8. # start the ssh-agent in the background

$ eval `ssh-agent -s`

> Agent pid 59566

c. $ ssh-add ~/.ssh/id\_ed25519

1. Copy the SSH public key to GITHUB.

$ clip < ~/.ssh/id\_ed25519.pub

# Copies the contents of the id\_ed25519.pub file to your clipboard

1. In the upper-right corner of GitHub page, click your profile photo, then click Settings.
2. In the user settings sidebar, click SSH and GPG keys.
3. Click New SSH key or Add SSH key.
4. In the "Title" field, add a descriptive label for the new key. For example, if you're using a personal Mac, you might call this key "Personal MacBook Air".
5. Paste your key into the "Key" field.3.
6. To test the connection

$ ssh -T git@github.com

If you get any error like Permission denied (publickey) error when using Git make sure ssh is running from which path

Make sure your SSH is registered to correct path by reregistering it.

1. $ eval `ssh-agent -s`
   * Agent pid 59566
2. $ ssh-add ~/../Srujana/.ssh/id\_rsa

**Project Setup**

1. Initialize Git:

git init

1. Add files to the Staging Area for commit:

git add .

git add README.md # To add a specific file

1. Before we commit let’s see what files are staged:

$ git status # Lists all new or modified files to be committed

1. Commit Changes you made to your Git Repo:

git commit -m "First commit"# The message in the " " is given so that the other users can read the message and see what changes you made

1. Uncommit Changes you just made to your Git Repo:

git reset HEAD~1# Remove the most recent commit# Commit again!

1. Add a remote origin and Push:

git remote add origin remote\_repository\_URL# sets the new remote

or

git remote set-url origin g[it@github.com:SRUJANA-P](mailto:it@github.com:SRUJANA-P)ENUGONDA13/AffliateWorld.git

git remote add origin [git@github.com:SRUJANA-](mailto:git@github.com:SRUJANA-)PENUGONDA13/simpleMail.git

1. git remote -v
2. git push -u origin master # pushes changes to origin
3. See the Changes you made to your file:
4. git diff # To show the files changes not yet staged
5. Revert back to the last committed version to the Git Repo:

git checkout

$ git checkout -- <filename>

1. View Commit History:

git log

Git Clone

1. git reset HEAD --hard # To remove all not committed changes!

git clean -fd # To remove all untracked (non-git) files and folders!

**Git Commit Messages**

1. feat - a new feature
2. fix - a bug fix
3. docs - changes in documentation
4. style - everything related to styling
5. refactor - code changes that neither fixes a bug or adds a feature
6. test - everything related to testing
7. chore - updating build tasks, package manager configs, etc

**Deploying into heroku**

1. Setting up ssh

heroku keys:add ~/../Srujana/.ssh/id\_rsa.pub

1. Creating app

heroku create srujana-penugonda

1. Deploying to heroku

git remote

git push heroku master

**Setting environment variables**

We must set environment (config) variables before deploying application into heroku

1. heroku config:set key=value

2. heroku config:unset key=value