Study Notes for
Interview Preparation

by SahebCSE

Node JS

Complete Handwritten Notes

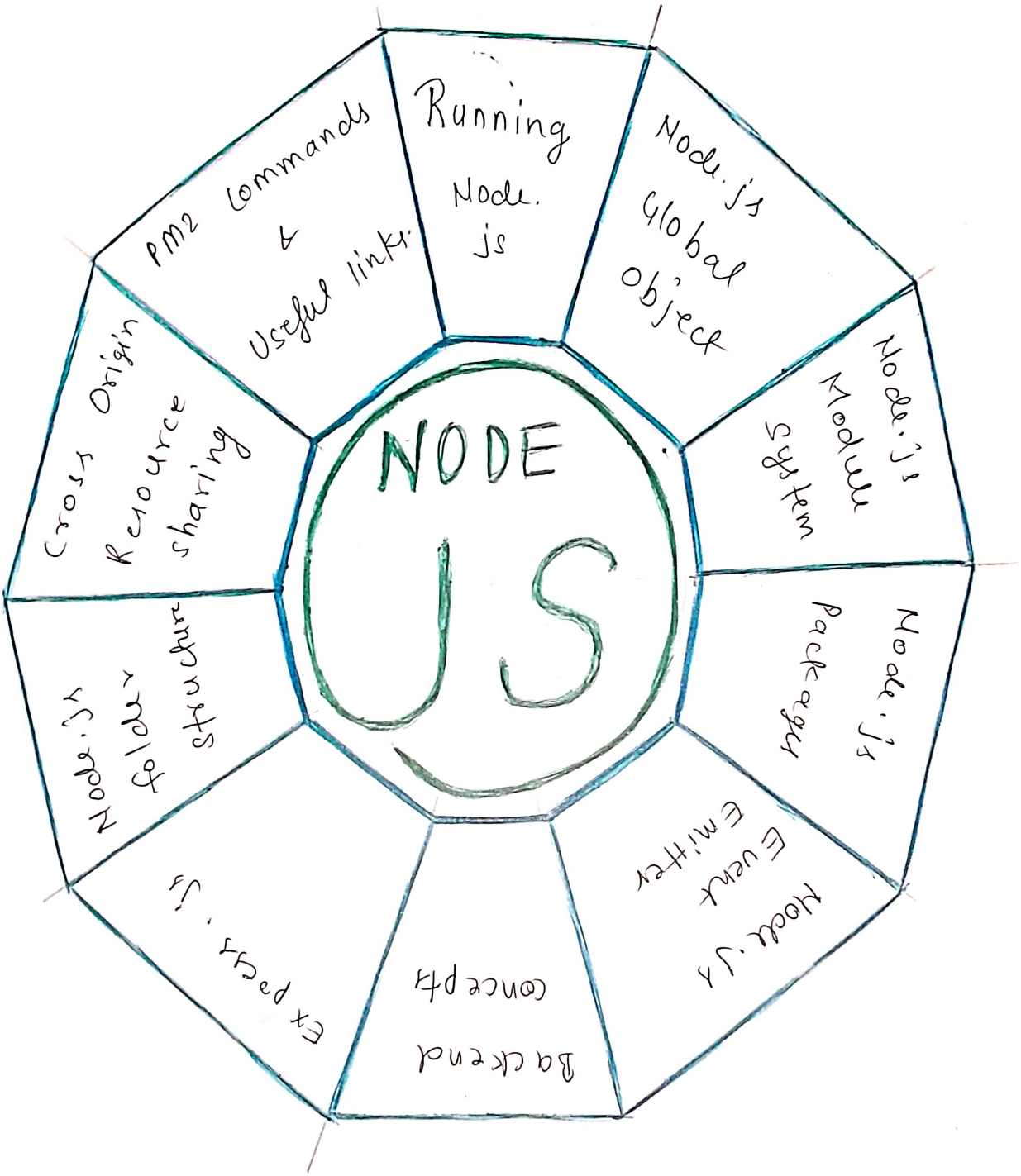
PLEASE SHARE AND HELP OTHERS 

@PrepTrain

@SahebCSE

PrepTrain

Do Join telegram.me/PrepTrain Community for latest Internships and Job Opportunites



running Node.js

For running Node.js

command	comments
node	Run the Node REPL in your terminal
node -version	Print your current Node version
node filename.js	Execute the Node code in filename.js

Node.js Global Object :-

In Node, we have a global object that we have always acess. Features that we expect to be available everywhere live in this global object.

e.g.

```
setTimeout(() => {
    console.log('hello');
}, 5000);
```

Node.js Module System :-

In Node.js each file is treated as a separate module. Modules provide us a way of re-using existing code.

The required function.

We can re-use existing code by using Node built-in require() function.

e.g

```
const fs = require('fs');
fs.readFileSync('hello.txt');
```

Built in Modules :-

Key built-in modules include:

- fs - read and write files on your file system
- path - combine path regardless of which OS you're using
- process :- Information about the currently running process, e.g. process.argv for arguments passed in or process.env created HTTP servers. for environment variables

ECMA Script Modules :-

The imports above use a syntax known as Common JS (CJS) module. Node treats Java Script code as CommonJS module by default. More recently, you may have seen the ECMA Script module (ESM) syntax. This is the syntax that is used by Type Script.

```
// In src/fileModule.mjs
function read(filename) { }
function write(filename, data) { }

export {
    read,
    write,
};
```

```
// In src/sayHello.mjs
import { write } from './response.mjs';
write('hello.txt', 'Hello world');
```

We tell Node to treat JavaScript code as an ECMA Script module by using the .mjs file extension.

- http - make requests and create HTTP servers
- https - work with secure HTTP servers using SSL / TLS
- events - work with the EventEmitter
- crypto - cryptography tools like encryption and hashing

Creating Modules :-

```
// In src/fileModule.js
function read(filename) { }
function write(filename, data) { }

module.exports = {
    read,
    write,
};
// In scr/sayHello.js
const {write} = require('./fileModule.js')
write('hello.txt', 'Hello world');
```

Node.js Event Emitter :-

Node.js provides a built-in module to work with events.

```
const EventEmitter = require('events');
const celebrity = new EventEmitter();

celebrity.on('sucess', () => {
    console.log('Congratulations! You are the best');
});

celebrity.emit('sucess'); // logs sucess message
celebrity.emit('failure'); // logs nothing
```

Many features of Node are modlled with the EventEmitter class. Some examples include the currently running Node process, a running HTTP server, and web sockets. They all emit events that can then be listened for using a listener function like on()

e.g.

```
const process = require('process')

process.on('exit', (code) => {
    console.log('About to exit with code: ${code}');
});
```

package.json :-

Most Node applications we create indu
a package.json file which means our Nod
applications are also Node packages.

The package.json file contains!

1. Name, version, description, license of the current package.
2. Scripts to automate tasks like starting, testing, and installing the current packa
3. Lists of dependencies that are required to be installed by the current page.

node_modules :-

when you run npm install the packages listed as dependencies in your package.json are downloaded from the NPM registry and put in the node_modules folder.

package-lock.json :-

The package-lock.json file is automatically created by NPM to track the exact version of packages that are installed in your node_modules folder.

Node.js Packages :-

Node developers often publicly share packages, that other developers can use to help solve common problems.

A package is a collection of Node module along with a package.json file describing the package

To work with Node package we use NPM. NPM includes two things :

1. The NPM registry with a massive collection of Node packages for us to use.
2. The NPM tool that you installed when you installed Node.

NPM Commands :-

Command	Comments
npm start	Execute the current Node package defined by package.json. Defaults to executing node server.js
npm init	Initialize a fresh package.json file

npm init -y	initialize a fresh package.json file, accepting all default opt Equivalent to npm init -yes
npm install	Equivalent to npm i
npm install <package>	Install a package from the NPM registry at www.npmjs.com Equivalent to npm i <package>
npm install -D <package>	Install a package as a developmer dependency Equivalent to npm install --save-dev <package>
npm install -g <package>	Install a package globally.
npm update <package>	Update an already installed package Equivalent to npm up <package>
npm uninstall <package>	Uninstall a package from your node_modules/ folder Equivalent to npm un <package>
npm outdated	check for outdated package dependencies
npm audit	check for security vulnerabilities in package dependencies
npm audit fix	Try to fix any security vulnerabilities by automatically updating vulnerable packages.

Backend Concepts :-

Client-server architecture :-

Your frontend is usually the client.

Your backend is usually the server.

In a client-server architecture, clients get access to data (or "resources") from the server.

API :-

Short for Application Programming Inter-face.

This is the set of functions or operations that your backend server supports. The frontend interacts with the backend by using these operations.

CRUD :-

Short for Create Read Update and Delete.

These are the basic operations that every API supports on collections of data.

RESTful :-

RESTful APIs are those that follow certain constrains. These include:

- Client-server architecture. Clients get access to resources from the server using the HTTP protocol.
- Stateless communication.
- Cacheable. The stateless communication makes caching easier.

CRUD Operation	HTTP method	Example.
Create	Post	POST / cards Save a new card to the cards collection
Read	GET	GET / cards Get the whole cards collection.
Update	PUT	PUT / cards / : cardId Update an Individual card
Delete	DELETE	DELETE / cards / : cardId Delete an individual card or mor rarely

xpress.js :-

GET Routes :-

```
// Get a whole collection of JSON objects
app.get("/cards", (req, res) => {
    return res.json(cards);
});
// Get a specific item in a collection by ID
app.get("/cards/:cardId", (req, res) => {
return res.json(cards[cardId]);
});
```

POST Routes :-

```
app.post("/cards", (req, res) => {
  // Get body from the request
  const card = req.body;
  // validate the body
  if (!card.value || !card.suit) {
    return res.status(400).json({
      error: 'Missing required card property'
    });
  }
  // Update your collection
  cards.push(card);
  // send saved object in the response to
    verify
```

```
    return res.json(card);
});
```

Routers :-

```
// In src/cards.router.js
const cardsRouter = express.Router();

cardsRouter.get("/", (req, res) => {
    return res.json(cards);
});

// In src/api.js
const cardsRouter = require('./cards.router');

const api = express.Router();
api.use('/cards', cardsRouter);
```

Node.js Folder Structure :-

One typical folder structure for an API following RESTful architecture and using the Express framework can be found below. Node servers typically follow the Model View Controller pattern. Models live together in one folder. Controllers are grouped together based on which feature or collection they are related to.

M2 Commands :-

M2 is a tool we use to create and manage Node.js clusters. It allows us to create clusters of process, to manage those process in production and keep our applications running forever. We can install the PM2 tool globally using `npm install -g pm2`

Command	Comments
☆ pm2 list	List the status of all process managed by PM2
☆ pm2 start server.js -i 4	Start server.js in cluster mode with 4 processes.
☆ pm2 start server.js -i 0	Start server.js in cluster mode with the maximum number of process to take full advantage of your CPU
☆ pm2 logs	Show logs from all process
☆ pm2 logs --lines 200	Show older logs up to 200 lines long.
☆ pm2 delete all	Remove all processes from PM2's list.

Cross Origin Resource Sharing

Something all web developers soon come across is Cross Origin Resource Sharing (CORS)

Browsers follow the Same Origin Policy (SOP), which prevents requests being made across different origins. This is designed to stop malicious servers from stealing information that doesn't belong to them.

Cross Origin Resource Sharing (CORS) allows us to allow or whitelist other origins that we trust, so that we can make requests to servers that don't belong to us.

For example:- with CORS set up properly https://www.mydomain.com could make a POST request to https://www.yourdomain.com