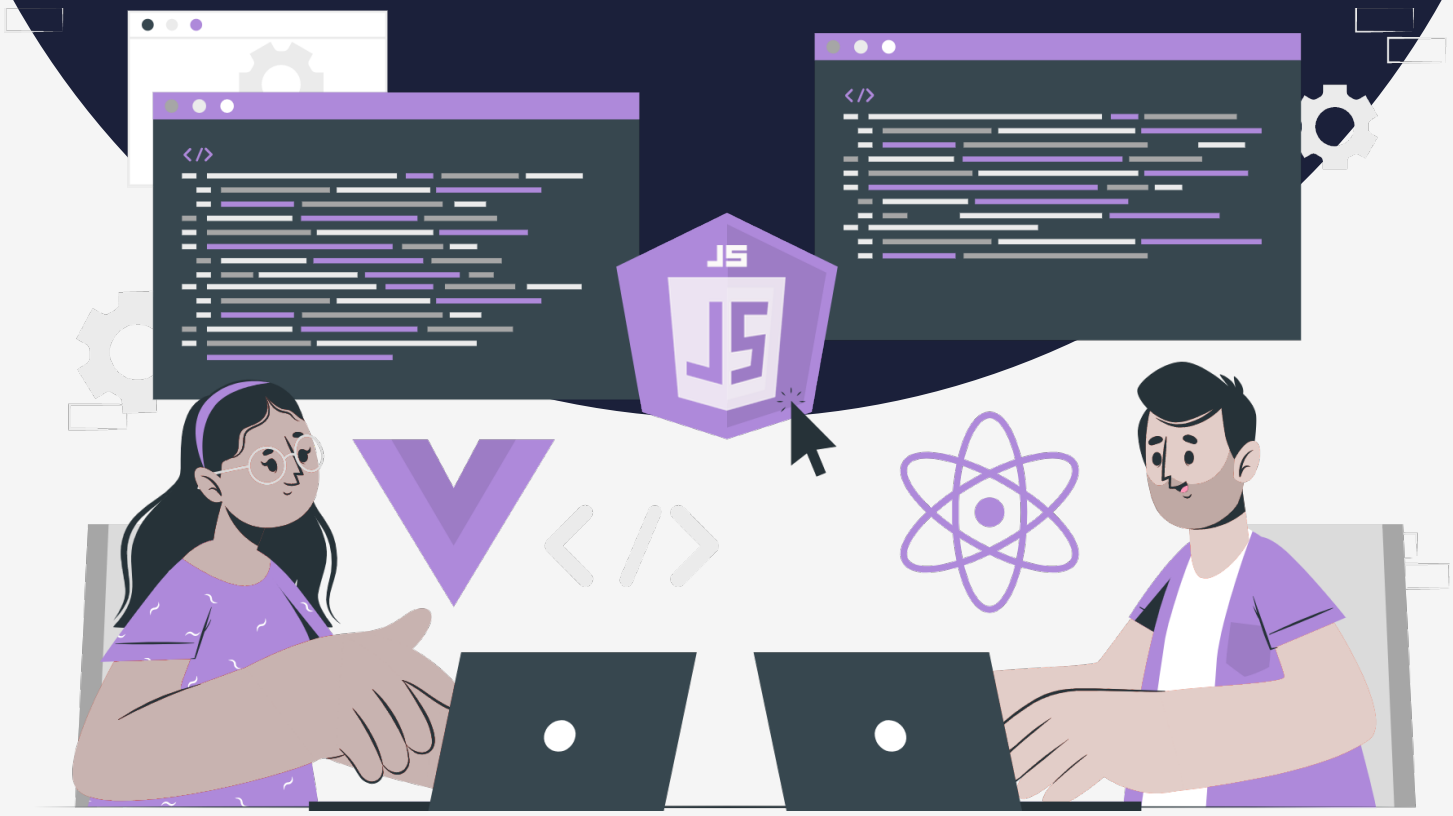
Lesson:

npm init and package.json



PW SKILLS

List of content:

NPM (Node Package Manager) is a package manager for Node.js applications. It is a **command-line utility** that makes it easy to install, manage, and share packages of Node.js code. NPM is the default package manager for Node.js, and it allows developers to easily install and manage dependencies for their Node.js projects.

NPM provides a central repository of packages, which developers can browse and search to find the packages they need. Packages can also be published to the repository, making it easy for other developers to discover and use them.

npm-init

The npm init command is used to initialize a new or existing npm package. It creates a package.json file that contains important information about the package, such as its name, version, dependencies, and other metadata.

Some examples of using npm init command are:

1. To create a new React-based project (React is a popular front-end library for building user interfaces, developed by Facebook. It allows developers to create reusable UI components and manage the state of those components in an efficient way) using create-react-app:

```
$ npm init react-app my-react-app
```

2. create-esm is a command-line tool that allows developers to create an ESM (ECMAScript Module) compatible package. ESM is a standardized module system for JavaScript that provides a more modern and efficient way of importing and exporting code between different modules.
   To create a new esm-compatible package using create-esm:

```
$ mkdir my-esm-lib && cd my-esm-lib
$ npm init esm --yes
```

--yes here, automatically accepts all the default settings and creates a package.json file without any prompts. It saves you from having to manually enter information like the project name, version, and author.

3. To generate a plain old package.json using legacy init:

```
$ mkdir my-npm-pkg && cd my-npm-pkg
$ git init
$ npm init
```

4. To generate a package.json file without any user input:

```
$ npm init -y
```

5. To create a new workspace within a project:

```
$ npm init -w packages/a
```

6. To create a new React-based workspace within a project:

```
$ npm init -w packages/my-react-app react-app my-react-app
```

6. To create a new React-based workspace within a project:

Installing packages in node

To install packages in Node.js, you can use the Node Package Manager (npm) command-line tool. Here are the steps:

1. Open a terminal or command prompt window.

2. Navigate to the root directory of your Node.js project.
   Run the following command to install a package from the npm registry:

```
npm uninstall package_name
```

   Replace package_name with the name of the package you want to uninstall.

3. Press Enter to run the command. npm will remove the package and its dependencies from the node_modules directory of your project.

4. If you want to remove the package from the dependencies section of your package.json file, run the following command instead:

```
npm uninstall package_name --save
```

   This will remove the package from the dependencies section of your package.json file.

5. If you want to remove a development dependency, run the following command:

```
npm uninstall package_name --save-dev
```

   This will remove the package from the devDependencies section of your package.json file.

6. You can uninstall multiple packages at once by separating them with spaces in the npm uninstall command. For example:

```
npm uninstall package1 package2 package3
```

   This will uninstall package1, package2, and package3 from your project.

package.json

The package.json file serves as the centerpiece of the Node.js system and is considered the project's manifest file, containing crucial metadata information. Understanding and learning how to work with this file is a fundamental aspect of working with Node.js. It serves as the first step towards becoming proficient in Node.js development.

To be more specific, the metadata information in the package.json file can be divided into two categories. The first category is **identifying metadata properties**, which includes details such as the project's name, current version, author, license, and project description. The second category is **functional metadata properties**, which includes values related to the project's functionality, such as the entry or starting point of the module, project dependencies, scripts being used, and repository links for the Node.js project.

Creating a package.json file:

package.json file can be created in two ways:

1. **Using npm init :** With this command, system expects user to fill the required information. It provides with default values which are editable by the user.

Syntax:

```
npm init
```

2. **Writing directly to file :** You can also directly write into file with all the required information and can include it in your Node project.

Example: A demo package.json file with the required information.

```
{
  "name": "PWSkills",
  "version": "1.0.0",
  "description": "Skill Channel",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node start.js",
  },
  "engines": {
    "node": ">=7.6.0",
    "npm": ">=4.1.2"
```

```
  },
  "author": "PWSkills",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.17.1",
    "express": "^4.15.2",
    "express-validator": "^3.1.2",
    "mongoose": "^4.8.7",
    "nodemon": "^1.14.12",
  },
  "devDependencies": {},
  "repository": {
    "type": "git",
    "url": "https://github.com"  //you can put a sample git repo url
  },
  "bugs": {
    "url": "https://github.com/"
  },
  "homepage": "https://github.com/"
}
```

Explanation:

name: This gives the name of application/project.
version: This gives the version of your application(version must follow semantic versioning rules)
description: This gives the description about the application, purpose of the application, technology used like React, MongoDB, etc.
main: This gives the entry point of the your app. It specifies the main file of the application that triggers when the application starts. Application can be started using npm start.
scripts: The scripts which needs to be included in the application to run properly.
engines: The versions of the node and npm used. These versions are specified in case the application is deployed on cloud like heroku or google-cloud.
keywords: Specifies the array of strings that characterizes the application.
author: It consist of essential information about the author like name, email and other author related information.
license: Specifies license to which the application confirms are mentioned in this key-value pair.
dependencies: The third party package or modules installed using npm are specified in this segment.
devDependencies: The dependencies that are used only in the development part of the application are specified in this segment. These dependencies do not get rolled out when the application is in production stage.
repository: It contain the information about the type and url of the repository where the code of the application lives is mentioned here in this segment.
bugs: The url and email where the bugs in the application should be reported are mentioned in this segment.