Node Modules

Be Modular

Our goals

- Learn how to break our code to reusable chunks
- Install community packages using npm
- importing and exporting
- Best practices

What is Module

- Reusable piece of code that is placed in a single js file
- Some programming language require the shared libraries to be a class in JS it can be anything
- each file can choose what to export and make public
- 3 kind of modules
  - Core Modules (not a lot of those)
  - Third party modules - (need to install)
  - Local modules in my projects

Module - EX - Core Modules

- To be able to use a module we need to import it
- In node we import a module by using the require command
- In this EX we will use the fs which is a module containing methods to deal with the file system
- readFile / readFileSync are methods in the fs module that are used to read file
- use those methods to read a file and print it to the console
- when you require with no path it will look first if there is a core module that match.

3rd party modules

- Those are modules that do not ship with node
- We have to install them before we can use them
- To install them we use a package manager called npm

What is a package

- A package contains one or more modules
- A package has a name
- A package has a version: major.minor.patch
- When installing a package what we installed is determined by the name and the version
- A package contains a file called package.json with information about the package
  - name
  - version
  - other packages this package depends on
  - entry point file

What is NPM

- Node Package Manager
- Used to Install community packages
- Used to publish my packages
- Can be used to install private packages
- Can be used to publish private packages
- Maintaining the versions of the packages
- NPM is installed when we installed node
- The packages are hosted remotely
- Packages can be private or public
- We can create our own company npm which saves the company packages on S3

Install community package

- If we want to install a package we need to decide if we want to install the package globally or locally
- A global package is usually installed to add another command to our terminal
- We will mostly want to install the package locally
- a local package will be installed in a folder called node_modules
- The node_modules folder will not be pushed to our SVN
- To install a package we use the command
  - > npm install <package-name>
- To uninstall we use
  - > npm uninstall <package-name>

Install community package - EX

- Install a package called lodash
- import it using the require command
- the module contains a method called upperCase which we will use to print an upper case hello world
- when we require a package, node will first try to match it with the built in modules, then it will look for it in the node_modules recursively

Install package - Problem

- The node_modules folder is not pushed to SVN this means that if a team member clones our code and runs it he will get an error that lodash package is not found
- We want the packages and versions we installed to be written and when another team member clones the project that all the packages written in that file will be installed
- The information about our project and the dependencies of the project is written in the file package.json
- npm can help us generate that file with the command
  - > npm init
- So when a team member sees a package.json file he can install all the packages with the command
  - > npm install

package.json - EX

- Using the npm init command, create a package.json
- install the lodash package
- this should modify the dependencies in the package.json
- delete the node_modules folder
- run npm install and make sure the lodash package is still there
- Every local package we install has to be written in package.json

package.json - dependencies devDependencies

- There is different kinds of dependencies we can have on a package
  - dependencies (—save)
  - devDependencies (-D)
  - peerDependencies
  - optionalDependencies (—save-optional)

Building our own package

- Our package will contain a single module
- Our module will export a function that prints hello world on the screen
- the main key in the package.json determines the entry point and if non is written it will default to index.js
- after our package is complete we can publish it with
  - > npm publish
  - to do this you have to login first with > npm login
  - to login you will have to create an account in npm

Building our own package - module wrapper

- Before we start building our package we have to understand how node import mechanism works
- We mentioned that every file node considers as a module
- node wraps each file with a function
- The function has the following signature:
  - > function(exports, require, module, __filename, __dirname)
  - exports === module.exports - contains an object that will be exposed to whomever requires this module
  - require - used for importing module
  - module - contains the module object
  - __filename - the name of the file of the module
  - __dirname - absolute directory of the module
- This means that all our variables will be private to the module

Our Package - EX

- Now that we know how node is wrapping each module with a function, we can write our package and our module.
- We can use the exports object to expose our hello world function
- try to require our module and use the function we just wrote

Advanced NPM

- Understanding NPM and feeling comfortable working with it is a key ingredient for code reuse and micro service architecture
- We have to understand proper professional way of working with this tool as an organisation
- The following section will describe advanced features of npm and usage tips for larger development teams

NPM registry

- NPM registry is a REST server containing information about the packages
- When we install a package the package is identified by the name of the package and a version
- When we install a package npm will send a request to the registry and will ask information about the package and version
- That information will contain the URL where you can download the tar of the package
- npm will then open the tar and install the package in the node_modules
- By default the npm registry is set to: https://registry.npmjs.org
- It is possible to create your own npm that will store the packages on s3

NPM registry - commands

- View the registry that npm is currently set to:
  - > npm get registry
- To set npm to a different registry:
  - > npm set registry <registry-url>
- You can ask to install a package from a specific registry with:
  - > npm install <package-name> —registry <registry-url>
- You can publish a package to a specific registry with
  - > npm publish —registry <registry-url>
- You can set your own npm that will search for a package and if non is found will request the package from the community npm
- You npm can be private and available only for your team
- This will allow to publish private packages

NPM registry - EX

- Query the community npm for the package lodash and locate from the response the url of the tar of the package for version 0.1.0

Scoped packages

- Scoped package allows us to give a namespace to our packages
- That namespace will be reserved for our organization.
- For example Angular packages are named @angular/<some-name>
- The namespace starts with @ sign
- It will be installed in the node_modules in a folder named @<the-namespace>
  - for example angular packages will be located in the folder @angular in node_modules
- To require them we need to place the full name with the namespace
- We can use the scope packages to say to npm:
  - We want the scope packages to be private and the unscoped ones they can be public
  - We want the scoped packages to be pushed to a different registry then the public ones

NPM private package

- Every package we want to publish we have to decide if that package will be public and available for everyone to use, or private and available only for our organisation.
- private packages can be stored on our own npm, or we can create a private section in the npmjs community registry (will require us to pay npm)
- If we want to host private package on npm they have to be scoped
- Scoped packages will be default be private
- Before you can publish you have to login
  - > npm login —registry=<registry-url>
- If you create your own npm you can force users to register, you can limit users from opening an account and by doing so you can create private packages that are not scoped

.npmrc

- With .npmrc you can declaratively config you npm
- You can override configuration by typing in the file
  - configKey=value
- npm will look for configuration files in the following places
  - In the project folder
  - In the user home
  - global config file
  - npm built in config file
- It is recommended to place an .npmrc file per project
- When you type > npm login , an .npmrc file will be created in the user home with a token used for publishing so you won’t need to login every time

.npmrc - EX

- In the folder where we created our module before place an .npmrc
- We want to configure our project that package that are scoped we will push to our private registry with access set to restricted
- packages that are not scoped will be pushed to the public npm registry and set the access to public
- This file will guarantee that we won’t publish private packages to public, now we simply need to remember that our private packages are scoped
- Common mistake is to add .npmrc file in the project with the token. Never include secret tokens in your repository

package restricted

- If we created a private scoped package we need to allow other users to access our package
- You can do so when you log in to npm in profile settings
- or you can do it with the cli
  - > npm access grant read-only ywarezk @nz/stam

Do it yourself - create an npm

- I recommend organisations to have their own private npm
  - Save money
  - Faster
  - private package scoped or not
  - private read access
  - Really easy to do
- You can open private npm with:
  - npm-register
  - verdaccio
  - sinopia

Nerdeez private npm configuration

- We used npm-register
- We restrict read and write access
- Users can’t open an account this means all the packages inside are restricted
- packages are published to S3 - this means high availability and fast
- npm-register is used with the docker image and containers are duplicated for high availability
- packages are installed from nerdeez npm and if not found the request is proxied to the registry.npmjs.org and then brought from their and cached in nerdeez npm so the next time it will be installed from nerdeez npm.

private npm - EX

- Install the npm-register package and open a local npm registry
- try to publish to your private package
- try to make your registry save the packages in s3

Summary - npm best practices

- Create your organisation npm
- create a simple solution where your team can simply publish private and public packages
- use .npmrc per project to help you in your solution