Labs NodeJS Modules and Packages

Preparations:
This lab requires a working installation of Chrome, NodeJS and Git. These tools can be downloaded from:
-
-

Exercise 1. Using Modules

Pre task steps

Download or clone the github repository from http://www.github.com/johngorter/

Start a git command prompt and navigate to the cloned repository. Execute git checkout e657126. After this checkout, the subdirectory 'myserver' should be created.

Task 1. Inspect the project structure and open the file ./myserver/server.js in your editor.

We are going to use express as the package to serve html files. to the client. Write the code to require the express module and write the code to serve files from the current directory.

The code to require express is:

```
let express = require('express');
let app = express();
```

The code to return a file is:

```
app.get("*.html", (req, res) => res.sendFile(__dirname + req.url));
```

Run the nodejs server with the command
node index.js from the myserver folder where the index.js is located.

If all went well, you should be looking at a browser page serving the HTML page named index.html in the same directory. Inspect the source of this page. It uses socket.io to connect to the server and accepts data from the server to display in the black console at the bottom.

If all went wrong, issue command git checkout 1e712b5.

Run node myserver/index.js and check if the page displays in your browser. Inspect the solution and make sure you understand the project structure.

We have built a webserver that serves html content. The socket code is hardcoded in the content to communicate back to the server. We do not want the website author to be responsible for writing the socket code for communicating back to our server. We will fix this in the next exercise.

Exercise 2. Creating Modules

Pre task steps

Download or clone the github repository from http://www.github.com/johngorter/

Start a git command prompt and navigate to the cloned repository. Execute git checkout c263d8d. After this checkout, the subdirectory 'myserver', 'injector' and 'watchdog' should be created.

In this exercise you have to create the 'injector' and the 'watchdog' modules to complete the project.

Task 1. Create the injector module

The index.js in the 'myserver' directory should give enough information on how to expose the modules.

The injector module should expose an inject function that takes a file, a marker and another file that has the injector-code. The inject function should also take a last parameter callback that gets called after the inject function is done.

The inject function should read the file given in its first parameter, read the file given in the third parameter and replace the contents of the file at the place of the marker with the injector-code read. The inject function should call the callback function and provide the injected html back to the caller.

complete the code in index.js in the directory 'injector'.

Task 2. Create the watchdog module

The index.js in the 'myserver' directory should give enough information on how to expose the modules.

The watchdog module should expose a constructor function that watches a directory given as a first parameter. fs.Watch can be used to watch a specific file or directory for changes.

The watchdog module export should create objects that inherit from EventEmitter.

When a file in the directory changes, the watchdog should emit the 'refresh' event.

Task 3. Test the code

navigate into myserver directory and execute 'node index.js'. If all went well you should see a browser serving an html page with information. Test that when you change the source of the page in your editor, the browser automatically refreshes the page.

If all went wrong, issue command git checkout 7d06f2c.

Navigate to myserver and run 'node index.js'. Check if the page displays in your browser. Inspect the solution and make sure you understand the project structure.

The socket scripts are now injected into the page instead of written into the page. We have built a server that, when started, live reloads our html when building websites. However, we have to build our website in our myserver module, this is not out ultimate goal. We want our server to start from everywhere and serving and injecting code from every folder. We'll complete this step in the next exercise.

Exercise 3. Using and Creating Packages

Pre task steps

Download or clone the github repository from http://www.github.com/johngorter/

Start a git command prompt and navigate to the cloned repository. Execute git checkout f751341. After this checkout, the subdirectory 'myserver', 'injector' and 'watchdog' should be created.

Task 1. Generate package.json and install dependencies

In this exercise we are going to create a package out of our myserver module, install and reinstall all dependencies for this project. The myserver module needs express and socket.io to function.

Navigate to the myserver folder and create a package.json for this module. use npm to scaffold a package.json.

Make sure that a package.json is generated for this module.

Install the packages express and socket.io locally. Make sure the dependencies are saved in the package. Use the npm command for this action.

Check to see if the package.json lists the dependencies for these installed packages.

Test the project by running node server.js. Close the browser after testing.

Delete the subdirectory node_modules from the myserver folder.
Reinstall the node_modules using npm install.

Test the project by running npm start. Does it work? Why not? Fix this!

Task 2. Generate a CLI for myserver

Navigating to myserver to start the server is a pain. Having to host the index.html inside this module is also not a good solution.

Change the package.json file. Add a bin section that hosts the "my-server" command and Add the bin directory to the myserver directory. Create a file inside the bin folder that holds the shebang command and starts the server.

The contents of this file should be:

```
#!/usr/bin/env node
'use strict';
require('../server').start();
```

Save the file and in the folder myserver, issue the command 'sudo npm link' to make a symbolic link to this location.

Execute the command 'my-server' and make sure the command succeeds.

Task 3. Extract the website and use my-server as a tool

Create a new folder in the root of the project named ‘myapp’. If it went well you should have four folders:
- injector
- watchdog
- myserver
- myapp

Move the html page from myserver to myapp. The myserver folder should not have the index.html. This is now located in a whole new project folder.

Navigate to myapp and issue command ‘my-server’. If all went well the browser should display the index.html page. Any change to the index.html in your favorite browser wil reload the page.

Now, my-server is an external tool which can be used for website authoring. This CLI tool is built on NodeJS, makes use of filesystem watchers and websockets to streamline website authoring.

If all went wrong, issue command git checkout aa40a6e for the complete and working solution (on MAC).