Ex3: Internet 2015/2016

- Read node.js stuff
  - the tutorial: http://nodebeginner.org/ and http://www.devshed.com/c/a/JavaScript/JavaScript-Exception-Handling/ and http://net.tutsplus.com/tutorials/javascript-ajax/introduction-to-express/
  - Go over the docs: http://nodejs.org/api/
    - specifically http://nodejs.org/docs/latest/api/modules.html and http://nodejs.org/docs/latest/api/net.html and http://nodejs.org/api/fs.html
  - Watch the video: http://goo.gl/asGxZ
- And some HTTP Stuff
  - http://www.jmarshall.com/easy/http/
  - http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
  - Read about the static request part of this peregraph http://en.wikipedia.org/wiki/Web_server#Path_translation
- You must read and follow the JavaScript coding conventions: http://javascript.crockford.com/code.html

- In this exercise you will develop a **static HTTP(version 1.1) server module** on top of node.js. **Module** is node.js component that other developers can 'import' and use (in our case they will use it to start their own static HTTP server). **Static HTTP server** is a software that serves static resources (e.g. html, js, css, image files) as an HTTP responses to HTTP requests. Ex4 and Ex5 will get built on top of this Ex, your final goal would be to build an application that runs on top of your dynamic web server.
  - You should build a few node.js modules
    - The module 'hujiwebserver' that exposes:

- .start(port,rootFolder,callback(err)) returns serverObj which starts the server and upon readiness to accept HTTP requests execute the callback.
    - In case the server could not start it should execute the callback with your custom err object that contains the error reason.
    - serverObj should have a
        - .stop(callback) function that stops the server and call the callback once the server is down
        - Two read-only properties (Learn about Object.defineProperty() first - https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Object/defineProperty)
            - port property
            - rootFolder property
- The module 'hujinet' which is in charge of communication with the node 'net' module including receiving and sending HTTP calls. This module should 'understand' what part of the network stream is actually an HTTP message and then it should parse it.
- The module 'hujirequestparser' **<optional>** which exposes one method:
    - parse(string) return HttpRequest object.

- You can add any methods to any of those modules or additional modules.
- This is a basic way for a user to use your module:
    -
```
var hujiserver = require('./hujiwebserver');
hujiserver.start(80,'/some/Folder',function(e){
    e?(console.log(e)):(Console.log('server is up. port 80'));
});
```
- In addition you should add a folder that contains a test.js that uses the node http client to test your application. it should test it against your ex2. it should put ex2 in the root_folder and it should send requests that test if it got ex2 files. as well as error response upon wrong resource.
- it should support HTTP persistence (keep-alive mechanism):
    - Read: http://en.wikipedia.org/wiki/HTTP_persistent_connection and http://stackoverflow.com/questions/8403781/node-js-sending-binary-data-of-http-over-a-socket
    - Consider using code such as var fileAsAstream = fs.createReadStream(filePath); fileAsAstream.pipe(socket); to send a file data back to the network as the HTTP response body while reading it
    - you should only use the socket.write() function in order to write data without closing the connection(pay attention that you and the browser should use the content-length header in order to recognize the end of the body of the request/response).

        - socket.end() will close the connection.
        - You can assume that when an http request contains a body, it contains a content-length header that specifies the length of the body.
            - It means that you should close the connection only if either:
                - The version of the HTTP request is 1.0 and there is no 'Connection: Keep-Alive' header embedded in that request.
                - The request contains a 'Connection: close' header (you should close the connection only after completion of the response)
                - Timeout (2 seconds since the last data)
        - Your HTTP server should support the following content types (HTTP Response header):
            - *.js: JavaScript : application/javascript
            - *.tx: Text: text/plain
            - *.html: HTML: text/html
            - *.css: CSS: text/css
            - *.jpg: JPEG: image/jpeg
            - *.gif: GIF: image/gif
            - Any additional type of file that you have in Ex2 and does not appear in the list above (one can find the specific content-type here: http://en.wikipedia.org/wiki/Internet_media_type)
        - You should accept GET requests only. for any other type of HTTP request you should return HTTP response 500
        - Write a tool that will load this server(send as many concurrent requests as possible) in order and test the the result is as expected. (include this load.js file)
    - Security instructions
        - Make sure that the user can't get any files that are not under the root folder.
        - Make sure that if something bad happen to the processing of one request it won't crash the entire server. (DOS)
- Some instructions
    - You are allowed to create additional node modules and to have as many files as you wish.
    - You are not allowed to utilize the 'http' module, you should use the 'net' and the 'fs' module as your infrastructure.
    - You are not allowed to use any external node.js files nor plug-ins without asking us first.
    - Try to minimize the number of global javascript variables as possible.
    - Consider hoisting and write you code hoisted already
    - Keep in mind that this is a web-server, it should be able to serve thousands of concurrent requests. Pay attention to performance issues.
        - Don't do any I/O operation in a synchronized manner.
    - **You are allowed to do this exercise in pairs or solo.**

    - **Each group of students must submit once (one zip file)**
    - Compress all your files and submit fullName.ID9digists.ex3.zip
        - Add a partner.<partnerID-firstNameEng-lastNameEng/NoPartner).txt file to the zip
        - Add a readme.txt file to the zip the describes (1) What was hard in this ex? (2) What was fun in this ex? (We won't reduce points in case this part is empty) (3) What did you do in order to make your server efficient (4) Explain how did you test your server and include as details result as possible
    - **Submission date: 28/12/2015 23:55 pm**
    - We will reduce up to 10 points for each week of delay.
- Extra reading
    - http://en.wikipedia.org/wiki/Nodejs
    - http://en.wikipedia.org/wiki/C10k_problem
    - http://oreilly.com/openbook/webclient/ch03.html
- Good luck