Ex3: Internet 2013/2014

- 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 a web server that is a subset of the Express.JS web server (in addition to let it run a mail app that you will build and deploy it on Heroku).
  - You should build a node.js module. This module(filename: miniExpress.js) must

expose the following functionality:

- this is how your users should import and boost it.

```
var miniExpress = require('./miniExpress');
var app = miniExpress();
```

- the app object should expose the following methods:
  - .use(obj1,obj2) - for now it only allows to set the root directory that contains the static files via the following code:
    - app.use(rootResource,miniExpress.static(rootFolder));
    - it means that request to /rootResource/x/y/z will return a 200 OK response with the rootFolder/x/y/z resource as the body, and the right content-type and content-length
    - you should only support GET requests
    - Error handling:
      - In case you could not parse the HTTP request, return status 500 and explain (via the body) what went wrong
      - In case you got non GET request, return status 405 and explain (via the body) what went wrong
      - In case you can't find the requested resource return status 404 and explain (via the body) what went wrong
      -
  - .listen(port) - starts the web server which listens to port <port>
  - .close() - stops listening to new requests.
- Some impl' details:
  - upon receiving a message from the 'client'), assume it's an HTTP message and parse it, if it's not(an HTTP message) you should return an error response. In case that no new data has been received in the last two seconds you should end the connection.
  - 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
    - 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):
        - JavaScript : application/javascript
        - Text: text/plain
        - HTML: text/html
        - CSS: text/css
        - JPEG: image/jpeg
        - 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)
    -
- Add a tester.js that test your server, add documentation (doc.html) that explains exactly what you are testing (you should use the HTTP module in order to test)
        - you can publish your tester on the Students Forum
        - you are not allowed to copy and paste someone else tester
- Test your server as the web-server of your Ex2 files.
- 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.

- 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.
    - We will test this ex on Windows (Node.js V 0.10)

    - Try to minimize the number of global javascript variables as possible.
    - 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.
        - Try to write a tool that will load this server(send as many concurrent requests as possible) in order to test it. (include this load.js file)
    - **You are allowed to do this exercise in pairs**
    - **Each student must submit his/her own Ex3**
    - 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
        - Add your own Ex2 to the zip under "www" folder, make sure your server can serve it perfectly. (where 'www' is the root folder)
    - **Submission date: 15/12/2011 23:55 pm**
    - We will reduce up to 3 points for each day of delay.
        - Excluding weekends.
- Extra reading
    - http://en.wikipedia.org/wiki/Nodejs
    - http://en.wikipedia.org/wiki/C10k_problem
    - http://oreilly.com/openbook/webclient/ch03.html
- Good luck