COS301 Mini Project

General Guidelines for Group A

# Packages

We would prefer if from now on the middle level people pack and upload the packages to Gemfury since you work much closer with the implementation teams. If you want one of your team members to do this they should sign up for Gemfury and send an email to [renetteros@gmail.com](mailto:renetteros@gmail.com) with their username for me add them as collaborator.

* Please add additional info to your package files such as contributors, repository, keywords, description, files etc.:
  + <http://browsenpm.org/package.json>
  + <https://docs.npmjs.com/files/package.json>
  + If you prefer publishing your package via the command line you **must** have the publish config in your package file otherwise it will publish to the public npm index.
  + Do not remove any of the existing options as they are required.
  + Please do not publish packages that contain errors.
* To add dependencies automatically use ‘npm install package ***--save’*** in your package folder
  + Read <https://github.com/npm/node-semver> to make sure your dependencies are listed correctly
* To automatically change the version use npm version (<https://docs.npmjs.com/cli/version>)
  + MAJOR version when you make incompatible changes. (eg. Changed function signatures, changed exported functions, changed package to work with electrolyte)
  + MINOR version when you add new functionality in a backwards-compatible manner.
  + PATCH version when you make backwards-compatible bug fixes.
  + A version number is in the format major.minor.patch
* Use npm pack to package your file into a tarball.
* Upload the tarball to Gemfury.
* I suggest you also use tags on your git repo to keep track of the different versions, but that is optional.
  + If you use npm version inside a git repo it should automatically create a tag and version commit.
* Please create a readme file that can be packaged with your module.

# Electrolyte dependency injection

Since we need to use dependency injection here is a few guidelines we would like you follow:

* Go read <https://www.npmjs.com/package/electrolyte>
* Module.exports should be a factory function that returns an object (possibly the one that is currently exported)
* If you require the database module or any of the other group’s modules you should not use require but use exports[‘@require’] = [...] (after you have assigned the factory function to exports…)
* You can now remove these other COS301 module dependencies from your package.json
* If you want to make sure your package is a singleton use exports[‘@singleton’] = true
* If you have multiple files that have dependencies you can handle it in a similar way using a factory function that gets everything from the main file

# Package Code

* If your package contains more than one file, there must be a single ‘main’ file to serve as an entry point to the entire package.
* For trivial functions such as fetching stuff from the MongoDB I suggest you write your own functions and not depend on other modules
  + Consider naming your schemas in the format package\_schema to prevent duplicate schema names.
  + Try to put all of your schema declarations alone in a single js file (eg. schema.js) that your other files can just require.
* You do not need to maintain your package in the mid-level repos anymore – if you are not going to do it, you can delete the folder from the applicable repo after giving the newest packaged one to your groups
  + We made no changes except the added package file and sometimes an extra JavaScript file that serves as entry point to the package.
* To test your module ‘in working’ you can copy your work into the directory where your package is installed. (Somewhere inside the node\_modules folder)

# General Code

* Please go read the Google JavaScript style guide and try to apply its guidelines:  
  <https://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml>
  + Stuff regarding browser compatibility is obviously not that important since you are mostly writing server-side code.
  + Please pay special attention to the sections regarding JSDoc and document your code in that style (Thanks to the people who have already done it!)
* Please pay attention to Webstorm’s warnings – most of the time it gives constructive feedback.
  + You can greatly reduce the amount of irrelevant warnings if you enable the NodeJS library: While your project is open go to Settings‑> Languages & Frameworks>JavaScript‑>Libraries
  + Those little lines on the scrollbar – that’s the warnings.
  + If you are not using Webstorm yet, start using it. It is available for Windows, Mac and Linux and you can get a free license using your tuks email so you have no excuse.
* If you have problems with the JavaScript please come and talk to us (The top level people)
* If you need to use additional node modules, just make sure they are included in your dependencies
* Please get your stuff done before the deadlines!

# Views

* Each middle-manager/group should create their own frontend page(s) using handlebars and other technologies.
  + <http://handlebarsjs.com/>
  + <http://handlebarsjs.com/builtin_helpers.html> (ignore the part about the missing helpers)
  + <http://handlebarsjs.com/reference.html> (only the section on @data variables)
  + If you are struggling with handlebars, you can ask top-level for help.
* Talk to the integration teams if you have questions regarding how the pages should link to each other etc.
* Use index.hbs, thread\_example.hbs, test.hbs in the views folder as examples.
  + Your hbs file will be included at {{{body}}} in layout.hbs when it is rendered, so all JavaScript files and stylesheets included there will automatically be available in your file.
  + This includes **JQuery** so you can use it anywhere client-side
  + If a specific javascript file/library is needed for many pages, it can also be included there.
* index.hbs contains an example of how to create links dynamically based on the context and also how to iterate over an array.
  + Note the ‘**this’** keyword everywhere – it is required.
  + Note that you can also use handlebars in the inline JavaScript
  + Note the encodeURI function because the item is added to the URL
* thread\_example.hbs uses http GET arguments to place specific data on the page
  + The arguments (req.query.arg) are handled in index.js to fetch the context
* test.hbs uses client-side handlebars templates
  + This is only applicable if you will be fetching data using Ajax and need to change the html client side.
  + URL’S used only in ajax calls should be sent to a url starting with /ajax/ for clarity sake
  + Post calls can be handled in the routing js using *router.post()*
  + The templates are in the client\_side\_templates folder.
  + Compile your template using ‘handlebars filename.handlebars -mf name.js’ (Note that Handlebars must be globally installed (npm install handlebars –g) and copy the JavaScript file to public/javascripts
  + Include the JavaScript file in your document and use html = Handlebars.templates["client-template"](context); to render it.
* Add your pages to the routes
  + Create a separate JS file in the routes folder and require it in index
  + Send it the router object from index.js to this file and set your package’s routes in this folder.
    - If you don’t have a clue what I mean, just put your routes in index.js
  + You should **NEVER** create any sort of html in the routing JavaScript – it must be done in the handlebars template.
* AJAX
  + Your html components will have event-listeners (eg. onclick, onsubmit) **client-side** functions.
  + These functions must validate all content before sending it to the server. Input must be validated on the server as well sine client-side code can easily be bypassed.
  + Use $.ajax({ … }) to send the http requests.
  + Insert the returned data back into the DOM
* Please use **Bootstrap** in your files
  + <http://www.w3schools.com/bootstrap/>
  + <http://www.tutorialspoint.com/bootstrap/>
* If you are using any additional libraries please notify the integration people so we can include it in the final project

# Deadlines

The final demo is 17 April (The Friday after the recess). We want everything done by the previous Friday (10 April) to have time for final details and quality control.

The front-end part should already be done by Wednesday 8 April.