Digital BlackSmiths: Thutong LMS

Coding Standards Document

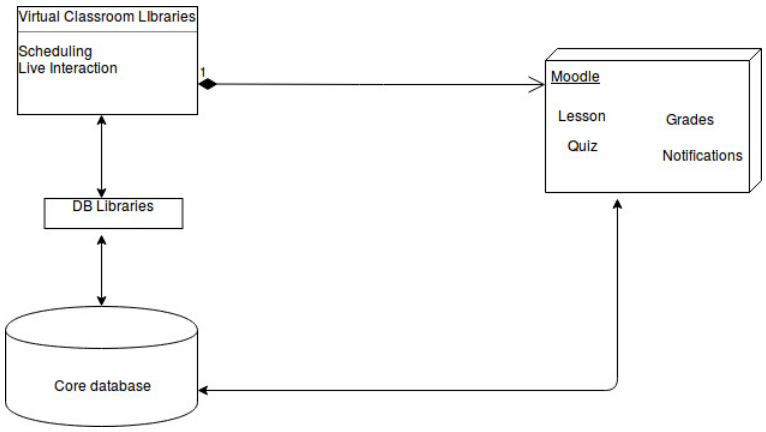
Group Members:

Fiwa Lekhulani
Daniel Rocha
Lebogang Ntlatleng
Lesego Mabe
Tlou Lebelo

Contents

This document contains the coding conventions created by the Digital Blacksmiths team, based on the various languages used to create the Thutong Online Learning System.
The team makes use of Moodle, Nodejs and WebRTC for the creation of the system.
Moodle is purely coded in PHP and coding standards are defined at https://docs.moodle.org/dev/Coding_style
The rest of this documentation will describe the coding conventions used for the JavaScript code wirtten.

1 Detailed System Design



2 Coding Conventions

To ensure that the code is flexible, readable, reliable and efficent the following aspects are considered and given the stipulated rules:

2.1 Naming Conventions

- Classes, functions and variables are to be named to descriptive as to what their purpose is.
- e.g var connectedUser, function createConnection(), etc.

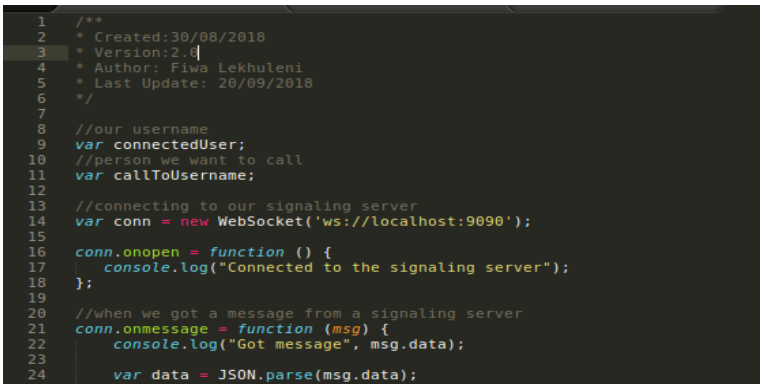
2.2 Layout Rules

The structure of each file must adhere to the following layout:

- Each file must have a file header wherein which the creation date, version, author and update history
  When a change is made to the file, the developer is required to change the date for last updated
- Only one class is allowed within a file, which may have infinite internal functions
- A class must have a description for its overall functionality and it's internal functions
- No more than one blank line is prohibited between lines of code, but with that said functions must be separated with a blank line for clarity and readability
- Every class definition, function and loop must be indented. The beginning brace must be placed in the same line as the definition of the loop or function but the ending brace must be in its own line followed an empty line.
- Code must be neat and self-explanatory, comments should be used when an action is fairly complex or needs additional instruction to another programmer.

2.3 Commenting Practices

- File header are composed of multi-line comments

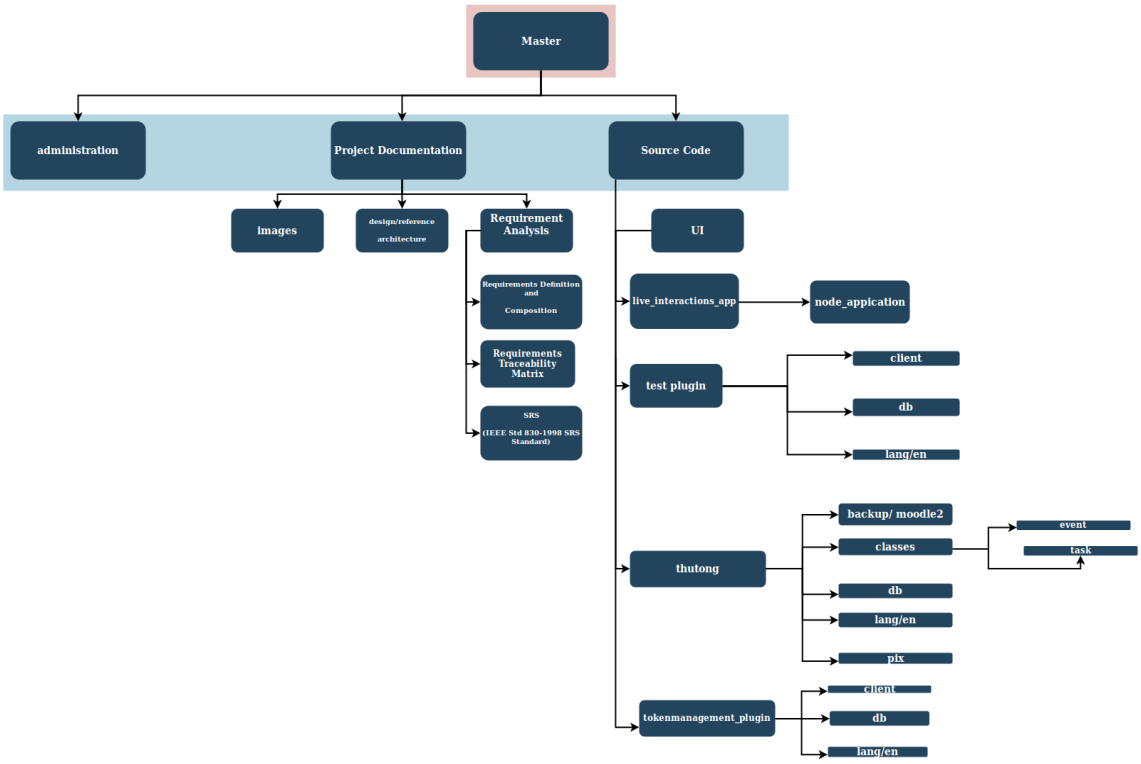


```
/**
* Created:30/08/2018
* Version:2.0
* Author: Fiwa Lekhuleni
* Last Update: 20/09/2018
*/

//our username
var connectedUser;
//person we want to call
var callToUsername;

//connecting to our signaling server
var conn = new WebSocket('ws://localhost:9090');

conn.onopen = function () {
   console.log("Connected to the signaling server");
};

//when we got a message from a signaling server
conn.onmessage = function (msg) {
    console.log("Got message", msg.data);

    var data = JSON.parse(msg.data);
```

- class or function definitions are made preceding the function using single line comments; these comments must be brief and solely give the core purpose of the class or function.
- Single line comments are to be used to give an explanation of what a specific line does; these should be only used when necessary. Avoid redundant commenting that can be easily understood from code naming conventions, methods and functionality.

```
//alias for sending JSON encoded messages
function send(message) {
   //attach our name to all messages sent
   if (teacherName) {
      message.name = teacherName;
   }

   conn.send(JSON.stringify(message));
};

//******************
//UI selectors block
//******************

var loginPage = document.querySelector('#loginPage');
var usernameInput = document.querySelector('#usernameInput');
var loginBtn = document.querySelector('#loginBtn');

var callPage = document.querySelector('#callPage');
var hangUpBtn = document.querySelector('#hangUpBtn');

var localVideo = document.querySelector('#localVideo');
var localCanvas = document.querySelector('#lccanvas2');
var remoteUser;

//set a stun server for the peer connections
var configuration = {
   "iceServers": [{ "url": "stun:stun2.1.google.com:19302" }]
};
//do not display the call page when we start
callPage.style.display = "none";

// Login when the remoteUser clicks the button
loginBtn.addEventListener("click", function (event) {
   teacherName = usernameInput.value;

   if (teacherName.length > 0) {
      send({
         type: "login",
         name: teacherName
      });
```

3 File Structure



The image above shows the file structure of the Digital BlackSmiths on GitHub. The folders are grouped according to functionality of the specific folders' contents i.e. files are grouped according to their functionality.

Administration items are grouped together, source code is grouped according to functionality or respective sub-system

Every folder and document is given a description of it's purpose upon uploading

4 Code Review Process

Each members' code uploaded to GitHub is reviewed after it is committed, to ensure that it complies with all above-mentioned standards and so that all group members understand the purpose of newly uploaded code for overall understanding of the system and newly added functionality.