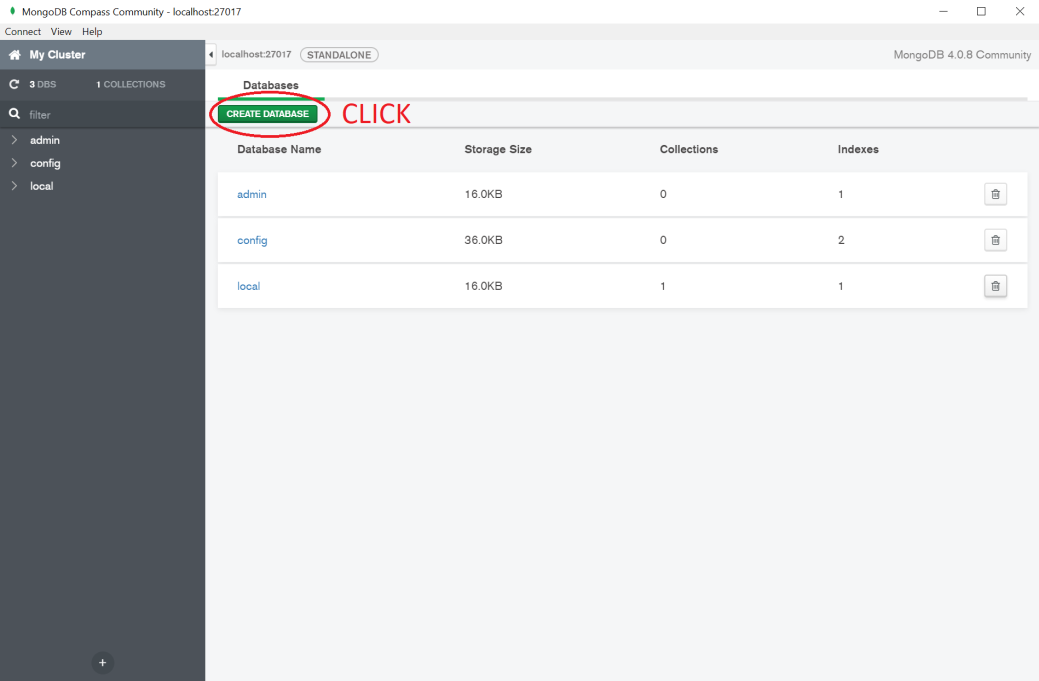
**Hangman Game Final Project**

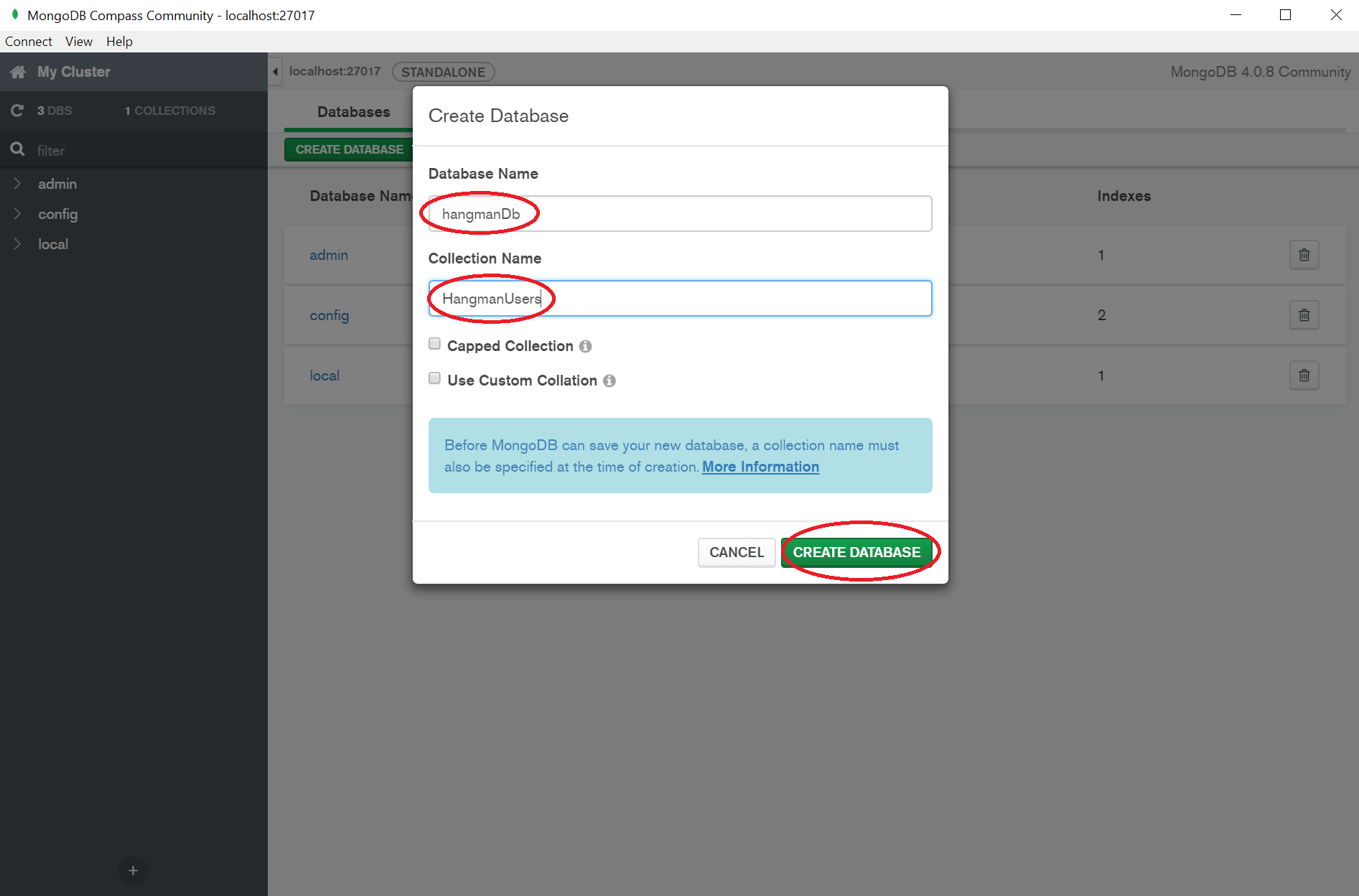
**By: Gergely Sajdik / 301142046 / ASB 9814 / April 9 @ 16:45**

**Introduction:**

For the final project I decided to create a Hangman game. This was the first assignment for me at SFU that has been open ended and allowed for creativity so I wanted to make a simple game since I haven’t had the chance. I personally enjoy all kinds of video games so my motivation was to be able to produce a game that is well known and can be played by almost anyone, while still being within reach of my current skill level as a programmer and time constraints. Considering all these factors I chose to make a Hangman game since it is known worldwide and has also been used as a tool for teaching vocabulary and spelling in grade school. The game is both relatable and useful so it seemed like a perfect fit for this project considering the skill level and time required to create it.

**Installation Instructions:**

1. Install node.js directly from the website
   1. <https://nodejs.org/en/download/>
2. Install MongoDB directly from the website (Version 4.0.8 was used with Win 10)
   1. <https://www.mongodb.com/download-center/community>
3. Create database in MongoDB Compass program
4. Name database “hangmanDb”, collection “HangmanUsers”, and click Create Database



1. Navigate to the “backend” folder of the project using cmd
2. Run “npm install” to make sure packages are installed
3. Run “node server.js” to set up local server through port 3000
4. Run index.html via browser of choice

**Project Description/Coding:**

The project is based on the world famous Hangman Game. The coding is done using vanilla JavaScript and utilizing node.js in order to create a front end and back end program layout (RESTful service). I have never had experience separating a game to allow the use of a back end server so I wanted to try it for this assignment in order to add a layer of difficulty and learn a new skill. Node.js is used for server side scripting and can generate dynamic page content, modify files on the server, collect form data, and modify data in a database.

I chose to use Node.js as a JavaScript server since it is easy to learn and offers great scalability. By adding additional nodes, applications can be scaled in a horizontal manner, and adding extra resources to single nodes is possible during the vertical scaling of the application. Learning Node.js takes less time since JavaScript is one of the most popular programming languages and easy to pick up due to the unlimited resources available online. Node.js has a large, active community of developers with many active repositories on GitHub. It is easily possible to find help or advice due to its popularity. Being written in JavaScript allows Node.js developers to write both front and back end in the same language using a runtime environment. There is no need to use any other programming languages which helps in testing and browser support (since almost all web browsers support JavaScript).

A main aspect of using Node.js was to allow the use of tokens and JSON format responses along with keeping data private and not accessible to users on the front end. Using the uuid (Unique Universal Identifier) library in Node.js it is possible to generate truly random or pseudo random numbers (practically unique). Whenever a user starts a session they are given a token (if they do not already have one), and the token is saved in the users cache and is provided to the Hangman game (server) automatically when it is launched. The token is used as a key to store a JSON object (which contains information about the user’s session and game status) in an array on the server. By having a unique token, it is possible to resume game sessions by using saved session data.

To access and install the correct libraries, NPM (Node Package Manager) was utilized to run the Hangman game. I ended up using express.js (standard for the majority of Node.js applications), cors, uuid (version 4), and random-words libraries/functions in this game.

As the user plays the game, the front end makes calls to the back end to create a random word, check passed letters for existence within the word, keep track of guesses made and allowed, as well as notifying the front end if the game is over. Since all the methods are executed in the back end, none of the data or methods are available to be publicly seen by front end users, which creates a layer of security and privatization. This is more commonly known as a RESTful API. The separation of the user interface concerns and data storage concerns improves the portability of the user interface across platforms as well as scalability. REST features a stateless client/server protocol which means that each HTTP has all the information needed to run it (no need to manage states). Objects in REST are always modified via URL, which is the identifier of each resource (End link or method) in the REST system. The URL allows access to information in order to modify it or share its exact location. By implementing a RESTful system, it offers additional decoupling for extreme scalability and it is not necessary for users (clients) to install anything.

MongoDB was used for some basic database implementation in order to keep track of a user’s stats. A token is saved locally to the users system and is used as the identifier key when searching the created database. I chose MongoDB since it is a NoSQL non-relational database, and because of this it can store structured, semi-structured, and unstructured data (dynamic schema). For this project the MongoDB database is a collection of JSON objects. It is possible to create multiple collections with each collection having a list of JSON objects, which offers flexibility compared to SQL databases that fixed/static defined schema.

**Experimental Results/Gameplay:**

At the start of the game a word is randomly chosen and the user tries to guess it by clicking on the letters. If the letter exists within the word, the blank where the letter should be is filled in, the guessed letter button is disabled, and the amount of guesses remaining stays the same. If the letter does not exist within the word then a strike is given by subtracting 1 guess from the amount of guesses left, the guessed buttons is disabled, and a part of the hangman is drawn. The guesser has a limited amount of letter guesses allowed. Should the guesser get all letters in the word before running out of guesses, they win the game – and should they run out of guesses then the game is over. When the game is over, regardless of a user win or loss, all letter buttons are disabled. When a new game is started all the buttons are re-enabled once more and the cycle repeats with the newly chosen word. If a user leaves in the middle of a game, they are given a loss upon their returning visit (updates as soon as they click Start Game on re-visit). If a user clicks on Refresh Stats, there stored statistics on the database will be displayed, and if they choose to Clear Sessions then their session and stats will be erased from the database (currently set to erase all collection contents).

Since this game is rather straightforward, there aren’t any notable experimental results to mention since the primary focus was on the gameplay and server implementation using Node.js.

***SIDE NOTE: I was able to find and fix the bug that I had mentioned to the TA during my presentation. The server now correctly updates upon clear sessions so no data is maintained in the database or the server, allowing for a full clean start upon clicking the Clear Sessions button.***

**Conclusion/Discussion:**

In conclusion I enjoyed working on this multimedia project. While it was not the flashiest or most relatable project possible for our course material, it was gratifying to work on an open-ended project for a change of pace. In my previous courses I was unable to learn and implement any back end programming since I was always responsible for front end and UI. Ironically I chose to focus on technical aspects and improving my back end knowledge, which I realize may not have been the best choice for a Multimedia based course so I’m hoping for some leniency in the marking if possible in that respect since I put so much effort into learning something new!

By making this game I was able to practice and re-familiarize myself on JavaScript programming and even expand on my previous knowledge by implementing a back end server and basic database. In the future there are some improvements I would like to implement to the game if time allows for it. I would like to add the possibility of loading in external text files as custom word banks for greater word possibilities and customization. Another important upgrade would be to the general UI and appeal of the game since it is barebones right now with most of the focus being on the technical features instead of the visual appeal!

**References:**

<https://www.mindinventory.com/blog/pros-and-cons-of-node-js-web-app-development/>

<https://blog.abelotech.com/posts/generate-unique-identifier-nodejs-javascript/>