**Simulation App**

**Table of Contents**

1. Purpose
2. Technologies and Resources
   1. NodeJS, Express, EJS, and Socket.io
   2. C++
   3. GYP and v8
   4. VueJS
   5. Webgl2
3. Future Features

**Purpose**

The purpose of this application is to provide a framework for exposing multiple different simulations to the web. The original version of this application was written in C++ using OpenGL inside of SDL, but it has since been ported to a standalone OpenGL implementation. The OpenGL implementation is the basis of this application.

**Technologies and Resources – NodeJS, Express, EJS, and Socket.io**

NodeJS was used to allow a full javascript we stack to be utilized, as NodeJS is known for its performance. Furthermore, since NodeJS utilizes the v8 engine, we can create C/C++ addons to allow for access to existing, mature code bases as well as fast, customized computations. This functionality made it so that the already existing simulation code would not have to be rewritten in a different language.

Express was the web framework of choice for this application over MeteorJS because MeteorJS comes with a lot of bloat in the form of features that are not needed for this application. Express is very lightweight, and yields a lot of space for customization.

EJS stands for effective javascript templating, and is a templating framework that renders html server side before it is sent to the client. This is useful for adding additional logic to templates before rendering them, and allows for the removal of redundant code.

Finally, socket.io was used because it is the most supported and robust web socket implementation for NodeJS at the time of writing this. This technology allows for real time communications between the client and server either the client can send data, or the server can push data to the client. When using web sockets, it is important to note that this data is being passed back and forth through a continuously open TCP/IP connection between the client and the server. AJAX cannot replace this functionality, as it is not able to enable the server to push data to clients, and it slows down the transfer of data because a new TCP/IP connection is made for each call. Do not attempt to replace web sockets with an AJAX solution.

<https://nodejs.org/en/>

<https://expressjs.com/>

<http://ejs.co/>

<https://socket.io/>

**Technologies and Resources – C++**

Because our original simulation was written in C++, it was kept to continue reaping the benefits of its speed. We are using a C++ addon for node to manage and run the simulations.

<http://www.cplusplus.com/>

**Technologies and Resources – GYP and v8**

GYP is a tool that is used to support building cross-platform C++ applications, similar to CMake. Since the official NodeJS documentation for C/C++ addons utilizes a plugin called node-gyp to create addons, GYP was utilized. With our C++ addons, the v8 engine is utilized to create javascript objects to return to the caller. It is worth noting that NodeJS, GYP, and the v8 engine were all developed by Google.

If the installation for node-gyp hangs for a long amount of time, default to using the manual installation method.

<https://gyp.gsrc.io/index.md>

<https://developers.google.com/v8/>

<https://github.com/nodejs/node-gyp>

**Technologies and Resources – VueJS**

VueJS is a javascript framework that implements the SAM (State Action Model) design pattern. This framework was chosen because it is easy to learn and use, and at the time of writing this, is faster than both Facebook’s ReactJS and Google’s Angular 2.

<https://vuejs.org/>

<https://www.infoq.com/articles/sam-lessons-learned-front-end-reactive-architectures>

**Technologies and Resources – Webgl2**

Webgl2 was used over Webgl because it provides more access to features that OpenGL currently has. This made it easier to convert the application from OpenGL to the web.

<https://www.khronos.org/registry/webgl/specs/latest/2.0/>

**Future Features**

Features planned for the future include:

1. Adding multiple types simulations.
2. Allowing multiple clients to view the same simulation via a simulation group.
3. Improving upon the old C++ code.
4. Removing xlib completely.
5. Wrapping up any TODO stubs in the application.
6. Camera Controls.
7. Fix issues with other web browsers.

**How to set up the Application**

1. Pull the application from <https://github.com/edwardsjohnmartin/avalanche>
2. Install NodeJS and NPM.
3. Ensure that Python 2.7 is installed on your machine and that the proper build tools for C++ are installed.
4. In the command line, run **npm install -g node-gyp**.
5. Now, in the ../simulation-app/ directory, attempt to run **npm install.** If this fails, run **node-gyp configure** and then **node-gyp build**. Then, retry **npm install**.
6. Now, there should be both a node\_modules folder and a build folder in the ../simulation-app/ directory.

After the project is set up, to rebuild the simulation, run the command **node-gyp rebuild** in the ../simulation-app/ directory. To run the application, use **node server**, and then the application will be running at localhost:3000. To allow changes to be made while the server is running, look into installing nodemon for node. If this is installed, simply run the application using **nodemon** in the command line.

**Last Update**

Zackary Hall – April 11, 2017

**Created**

Zackary Hall – April 11, 2017