Load Balancer

Description

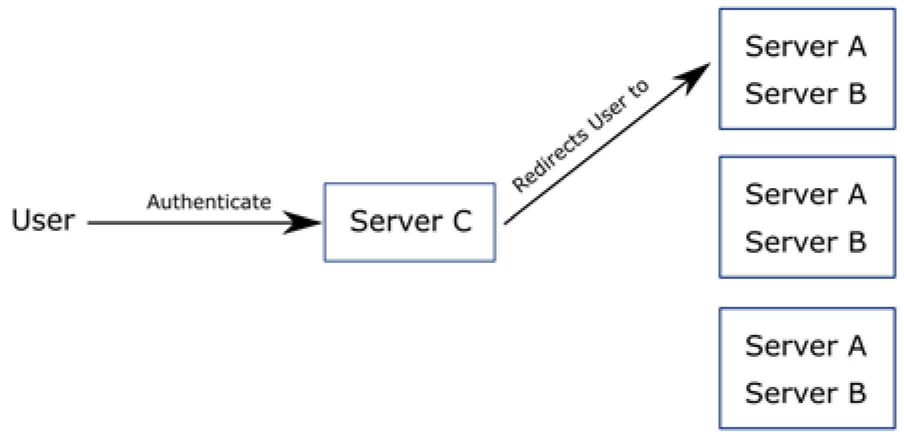


Figure 1: Implementation of the authentication and load balancer (Server C)

This is the load balancer and authentication server. The web service's domain refers to C as the entry point of all users. Users will then authenticate with C (by entering login information or creating a new account). After which C will redirect the client's browser to the IP address and port of the next available web-service package A and B (Processing server + Web application server), the order is determined by round-robin. After being redirected, the user will be able to use the web application normally. In addition, users can now have their session data saved by the server (based on their login username) so that they may continue working after logging out.

User login data is securely transferred from the client through HTTPS (SSL encryption). The password isn't stored, only the hash of the password is (with BCryptJS).

Project Structure:

app.js

Main entry point, servers initialization:

- Starts the Express JS server to serve the login UI (PUG) files to the browser
- Starts **(load_balancer.js**) the SocketIO server to listen for processing server registrations

lib/routes.js

Main Passport JS routes for authentication (to redirect the user to the correct PUG page such as login, failed login, success login), when successful login, redirect the client browser to the next available processing server

lib/load_balancer.js

Starts and maintain the Socket IO server to listen for:

- New Processing server registering their IP and port address
- Remove processing servers that disconnected

Processing Server

Overview

Designed as the server that handles dataset processing requests. Its main function is to store the user dataset into a workspace (designated to each user). It then receive requests from the user (through the web application) to process various data processing tasks that can be used for visualization, such as streamline tracing, surface tracing, or volume rendering.

It is a Node JS server that serves as the data processing service by running C++ visualization code with Node-Addon-API. After the web application is loaded to the client browser, the web app then connects using Socket IO to initiate a new user session. The user can then use the web application to upload 3D datasets to this server (along with the visualization task) and receive the corresponding visualization data for the web app to display.

Project Structure

app.js

- Starts socket/helpers/socket.js
- Starts a SocketIO client to connect to the Load balancer's SocketIO server to register the IP address and port of the processing server

socket/helpers/socket.js

- Starts a Socket IO server to listen for the React App's connection and all IO Events related to it. Such as:
    - Input Events: receiving datasets uploaded from the web app, or new/update processing tasks request
    - Output Events: sending resulting datasets, task results, or workspace snapshot back to the client

socket/helpers/task.js

Handles new or existing task input parameters. It receives the task parameters and perform the task (in c++ code, using node-addon-api), then it returns the task results. (This module does NOT handle IO events, only execute the task from the given parameters!)

socket/helpers/task/executeTask.js

This module is the SocketIO's INPUT event handler for task execute. Its performed when the web app sends a request (and parameters) to execute a task. This task can either be new or already existing. If task exists, it will modify the task's parameters and execute the task.

After executing the C++ task. It will send (hence OUTPUT) the results of the task (or error message) back to the client.

The resulting data of the task will be stored in the server also

socket/helpers/storage/samples.js

Holds a list of sample datasets and workspaces that the web app can request the processing server to send over

userdata/userdata.js

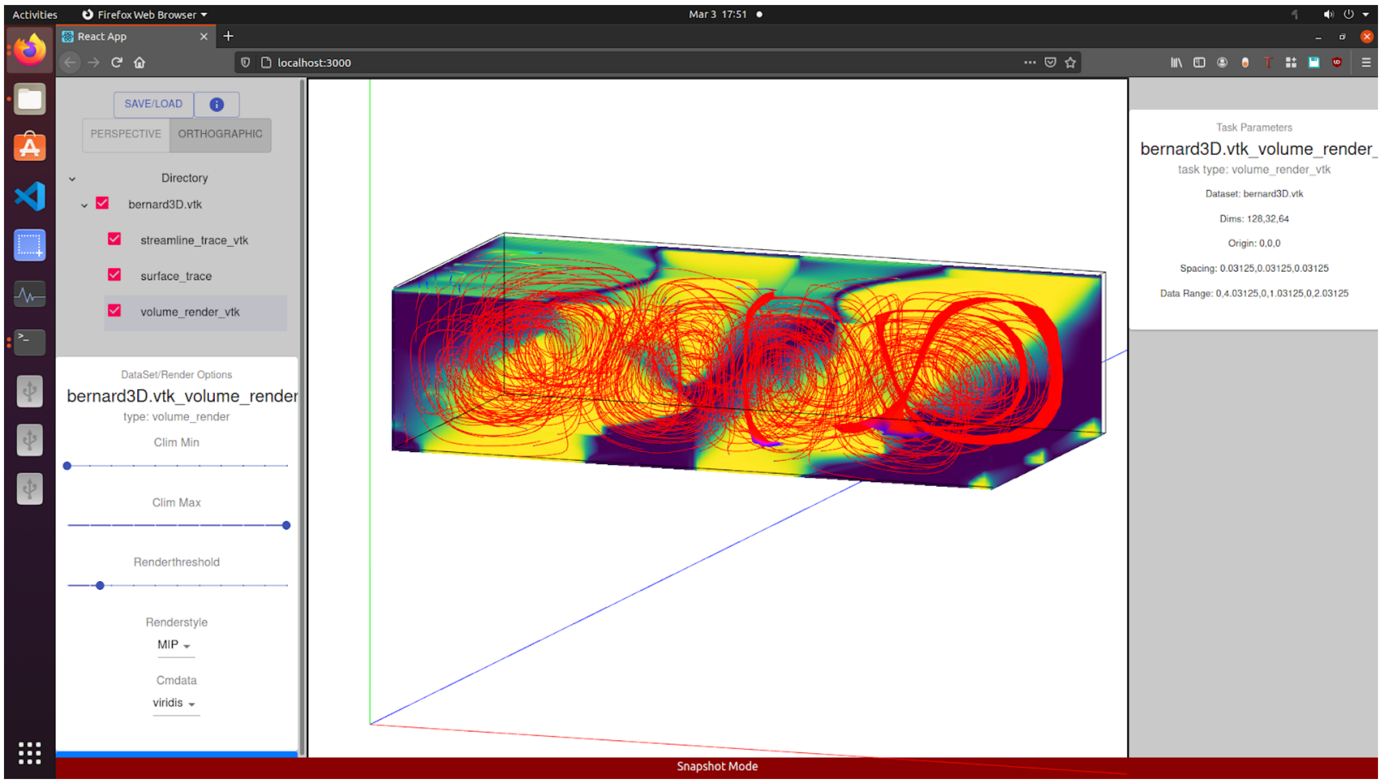
Handles saving and loading of ALL user data from and to disk:

- Saving/loading of workspace package (include both server and client data) to and from disk
- List all workspace owned by a user that is saved on this server

React JS Application Server

Overview

A NodeJS server that uses Express JS to serve a React JS web application. The client’s web application utilizes React JS to display user interface (UI) elements and facilitate user interaction. To display 3D visualizations, the client uses Three JS. There are future plans to add additional 3D and 2D visualization frameworks such as D3 to accommodate more visualization formats.



bernard3D.vtk_volume_render
type: volume_render
Renderstyle: MIP
Cmdata: viridis

Task Parameters
bernard3D.vtk_volume_render_
task type: volume_render_vtk
Dataset: bernard3D.vtk
Dims: 128,32,64
Origin: 0,0,0
Spacing: 0.03125,0.03125,0.03125
Data Range: 0,4.03125,0,1.03125,0,2.03125

Figure 1: A screenshot of the web app with a workspace snapshot loaded

Project Structure

all source codes are in /src

index.js

- Starts the Express JS server that will serve the React Application to the client's web browser
- Starts the SocketIO Client (./utilities/ioInstance.js) that will connect to the processing server for all communications
- Initialize all SocketIO input/output events

Once loaded to the client's browser, starts a SocketIO client connection to the processing server's SocketIO server