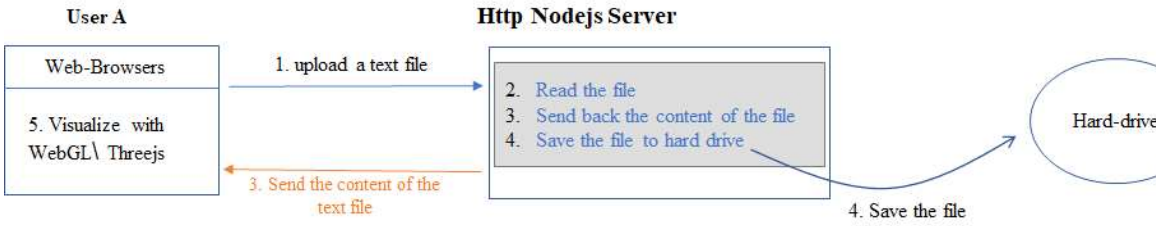
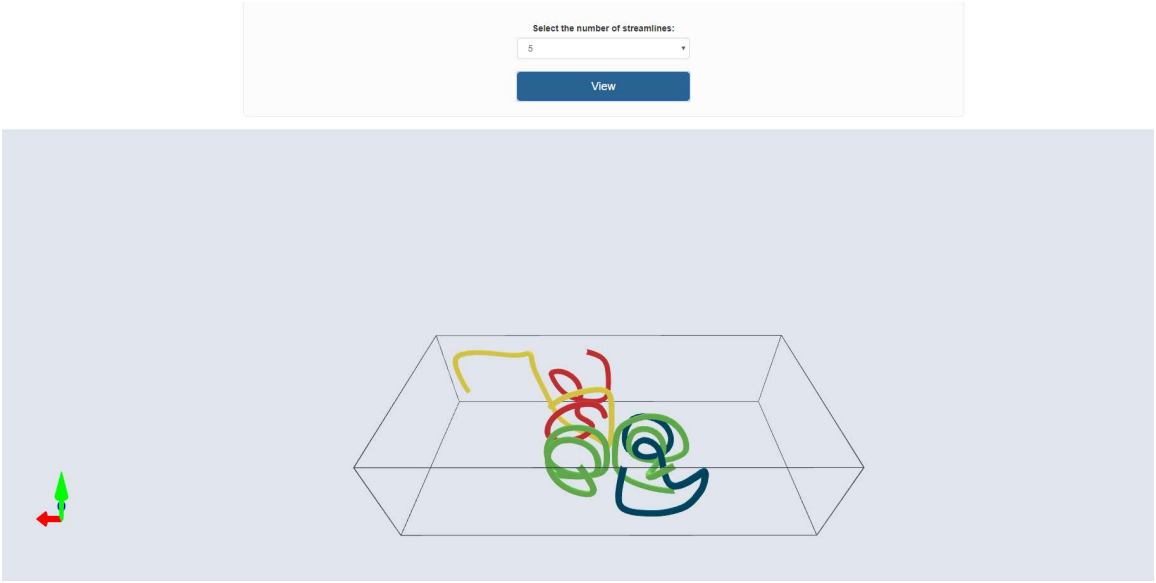
- **Project Description:**

Develop a web-based system so that a user can upload a text file which includes some 3D lines. After the server receives the text file, it sends back the content of the file to the front-end. In the meantime, the server also needs to save the uploaded file to the hard drive. The front-end will visualize the 3D lines by using WebGL/Threejs. The server needs to be implemented by using NodeJS.



Programming Language: Html, CSS, Javascript
Required Libraries: NodeJS for the server side, WebGL/ThreeJS for the front-end. Feel free to use Bootstrap/React or other CSS frameworks for designing the front-end UI

You can download the sample text file from this link: **streamline_points.txt**
This file has 5 lines, each line contains a set of 3D points separated by a space. The expected visualization of these lines as follows:


Select the number of streamlines:
5
View

- **Instructions**

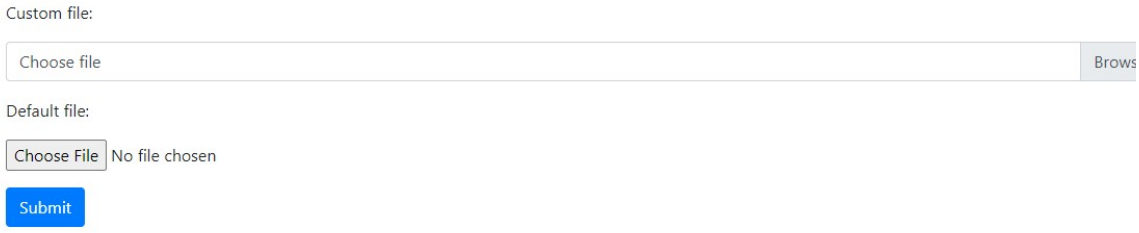
Here are some suggested instructions that you can follow:

Server Development:

Develop a HTTP server which receives a file and sends back the file content to the client by using NodeJS (or ExpressJS). Here are some good tutorial to start with:
https://www.w3schools.com/nodejs/nodejs_uploadfiles.asp
https://flaviocopes.com/express-forms-files/

Front-End Development:

Step 1: Create a simple HTML file which allows a user to select a file and upload to the server, then receive the response from the server.



Step 2: In the html file, add a visualization function with Threejs library. You can find a bunch of documentation and examples with ThreeJS at their official website:
https://threejs.org/
https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene

Try to visualize some 3D lines whose points are manually inputted. This might be a good example to test:
https://threejs.org/examples/webgl_lines_fat.html
Source: https://github.com/mrdoob/three.js/blob/dev/examples/webgl_lines_fat.html

Step 3: Combine Step 1 and Step 2
Once the client receives 3D lines from the server in Step1, uses the visualization function in Step 2 to visualize the lines.

Feel free to come up with your own solution for this project!

III. Bonus

After finishing the main features of the project, you can try to compress data during the transmission using gzip. Some sample tutorials are available here:

https://github.com/expressjs/compression
https://alligator.io/nodejs/compression/

Measure the page load time with and without using the data compression.

Standard Nodejs form upload is not reliable for large files, instead of using formidable, try using socket-io streams to upload the file instead. Here is a good tutorial: https://code.tutsplus.com/tutorials/how-to-create-a-resumable-video-uploader-in-nodejs--net-25445