DS 703 Geographic Information Systems
Data Visualization on Networks

Software Design Document

Deepshree Ravindran
IMT2012013

Haripriya Bendapudi
IMT2012018

Prateek Bansal
IMT2012032

Tarun Tater
IMT2012047

Instructor: Prof. S. Rajagopalan

Contents

List of Figures

1 Introduction

The purpose of this software design document is to provide a low-level description of "NetViz" - the network visualization application, providing clear understanding of the structure and design of each component of the application. This document gives the reader a good understanding of the working of the NetViz.

1.1 Goals and Objectives

The purpose of NetViz is to provide a platform using which, stock and flow data on any network can be visualized. A network consists of nodes, links and maybe spatial data related to the nodes. We propose to build a web application which, given a network (its topology) and data on the network (stock flow data), visualizes the data on the network. For example, in a road network with the bus stops taken as nodes and the connecting roads taken as edges, the number of buses waiting at a particular bus stop will be stock information and the number of buses moving on a road between two bus stops will be flow information for that edge. Therefore, given the co-ordinates of the bus stops and the stock and flow data related to the bus network, one can visualize it using NetViz.

The application also provides the user with the flexibility to change the shape/line-type used to represent a node/edge. The user also has the flexibility to choose how stock and flow are represented in the network.

Therefore, the final product must be fast, efficient and easy to use by even non-technical audience. It should provide a good amount of flexibility to the users without overwhelming them with too many options. The interfaces must be user friendly and intuitive.

1.2 Project Overview and Scope

Since NetViz is a web based application, it is composed of two major components: a client-side application and a server-side application. The main function of the client-side application is to take input from the user (topology, stock, flow data related to the network) and display the final visualization back to the user. It also provides options to the user to modify the visualization. The server-side of the application is for storing all the visualizations made by a particular user thereby allowing him/her to access it from any computer. Also, all computations related to any queries made by the user are done by the server-side of the application. The major system features can be divided into two categories - the core features, which are the essential functionalities that have to be provided by the application and additional features which just provide extra functionality.

1.3 Input File Format

- There should be 3 csv files in the following format(including headers) :
  - Nodes File :

id	Node Name	latitude	longitude	stock

All the columns are mandatory.

  - Edges File:

id	startNode	endNode	flow

All the columns are mandatory.

1.4 Core Features

1.4.1 User Login and Welcome

- Has a welcome message
- Requests for the user's id and password
- Has a "Register" option for unregistered users

1.4.2 Registration

- Asks the user to choose a username and password
- The username should be unique and the password has to be at least 8 characters
- On clicking save, the username and password get added to the database and the user is redirected to the login page

1.4.3 Data

This view allows the user to add the data that needs to be visualized. Once the data is added, it is displayed in a tabular format.

Nodes This tab shows details of nodes that have been added to the database. The main columns are:

- id
- name
- latitude
- longitude
- stock

The stock can be added along with the node data if the visualization that needs to be done is for the entire data.

Edges This tab shows details of edges in the network. The main columns are:

- id
- startNode
- endNode
- flow

1.4.4 Visualization

This tab shows the visualization of the data added. It also shows some statistics related to the data and provides methods for the user to change the settings of the data.

Network Visualization The visualization shows all the the nodes set in the correct topological order. It also shows the stock and flow data related to the network in terms of sizes of nodes, thickness of edges, color of nodes, etc.

Settings Here the user can choose how the stock and flow data need to be visualized (size of nodes, thickness of edges, weight of edge, etc). The user can also change the background color of the visualization window.

Statistics Some basic statistics related to the data such as number of nodes, number of edges, max and min values of stock and flow, etc are displayed.

1.5 Additional Features

1.5.1 File

New Project The user can start a new project for visualization.

Open The user can open an already existing database stored on the server and visualize the stock and flow of the network.

Save The database in use can be saved for future use.

Exit This option can be availed to close the application and exit from it.

1.6 Intended Audience

This document is a follow up to the Software Requirements Document (SRS). Although the SRS is meant for a more general audience, the design document is relevant mainly to those directly involved in the development of the NetViz application. This includes the team leads, database manager and software developers. The various sections of the document are modular. So, one can skip to the section that is relevant to them.

2 Data Design

2.1 Data Structure

The user uploads data in the form of a csv file. The file gets transferred to the server using an FTP protocol. The server then parses the file and stores the data on the database server (MongoDB) which can be on the same physical device or some remote device. Therefore, to access this database, we need a server-side scripting language. In this case, we choose to use MongoDB for backend database, AngularJS for front end, Express and NodeJS for server side system and D3 for visualization. All code is written using javascript. The data required for visualization is sent back to the client by the server in the form of a JSONs.

The client application will never access the database server directly. It always go through the application server. Also, all the data uploaded during a session is stored until the user deletes it. If the users selects the save option, the state of the visualization (bg-color, flow representation, etc) gets stored along with the data and the entire visualization gets recreated on re-opening the visualization.

2.2 Database Design

usersDB{
"_id" : string, unique,
"password" : string,
"data" : [{
"_id" : string, unique,
"nodeList" : [{
"nodeId" : integer, unique,
"nodeName" : string,
"lat" : double,
"long" : double,
"stock" : integer
},
...
...
...],
"edgeList" : [{
"edgeId" : integer, unique,
"node1" : integer,
"node2" : integer,
"flow" : integer
},
...
...
...],
"options" : {
"nodeColor" : string,
"edgeColor" : string,
"background" : string,
"flowRepresentation" : string
}},
...
...
...]}

Figure 1: Database Design

3 Architectural and Component-Level Design

3.1 System Structure

NetViz consists of two major components: a client-side application and a server-side application. On the server side, one can have two different servers if needed: an application server and a database server.

The client side application is a web page written using HTML, CSS and JavaScripit. It provides buttons, text boxes etc., which enable the user to interact with the application.

The server component receives data file, parses useful information and stores it in a centralized Mongo database. The state of a visualization can be stored by the server. This allows the user save a visualization and view the same visualization again at any later time.

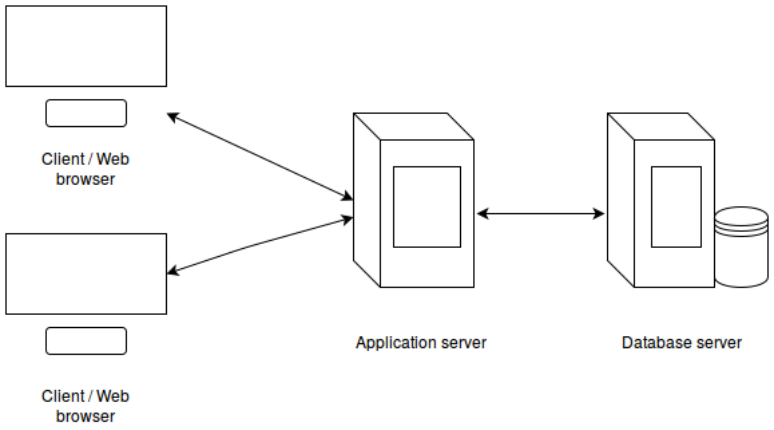

Figure 2: System Architecture

3.2 Node Class

Each node is an object of this class. The *node* class has the following attributes :

- node_id - (string) : a unique identifier for each node
- stock - (float) : the stock value accumulated at the node
- node_latitude - (double) : the latitude of the node
- node_longitude - (double) : the longitude of the node

Each instance of the *edge* class has two instances of node class. An instance of network class will have many nodes.

The *node* class has the getter and setter function for each of these attributes. The *node* class is referenced in the following solutions :

1. When an edge is added/deleted.
2. When flow data of an edge is updated.
3. When the node or edge representation is changed.
4. When the node shapes are changed in the network.

The class has no parent or child class.

3.3 Edge Class

Each edge is an object of this class. The *edge* class has the following attributes :

- edge_id - (string) : a unique identifier for each class
- starting_node - (string) : the id of the node which is the starting node for the edge.
- ending_node - (string) : the id of the node which is the ending node for the edge.
- flow - (float) : the flow on an edge.
- node_color - (string) : the color of the node.

The *edge* class has getter and setter functions for all the attributes. The edge class has no parent or child class. An instance of network has many instances of edges.

This class is referenced in the following situations :

1. When the edge representation is changed.
2. When the edge type is changed in the network.

3.4 Network Class

The Network class has the following attributes :

- nodeList - (array[node]) : an array having all the nodes present in the network.
- edgeList - (array[edge]) : an array having all the edges present in the network.
- nodeShape - (hashmap) : a hashmap from node type to shapes
- stockRepresentation - (string) : how the stock is represented (size of the node, color of the node, etc)
- flowRepresentation - (string) : how the flow is represented (thickness of the edge, color of the edge, etc)

The Network class also has no parent or child class.

This class has getter and setter functions for all the attributes. Apart from these, the following functions are also available :

- void addNode (node_id) : This method can be invoked when the user wants to add a node in the network. The method requires *node id* as input. The table "Node" is accessed in this method.
- void addEdge (edge_id) : When a user wants to add an edge, this method is invoked by giving an *edge id* as input. The table "Edge" is accessed in this method.
- node delNode (node_id) : A node can be deleted by this method by specifying the id of the node. The table "Node" is accessed in this method.
- edge delEdge (edge_id) : An edge can be deleted by this method by specifying the id of the edge. The table "Edge" is accessed in this method.
- void visualizeData() : Visualizes the data. The tables "Node" and "Edge" are accessed by this method.

3.5 UML Diagram

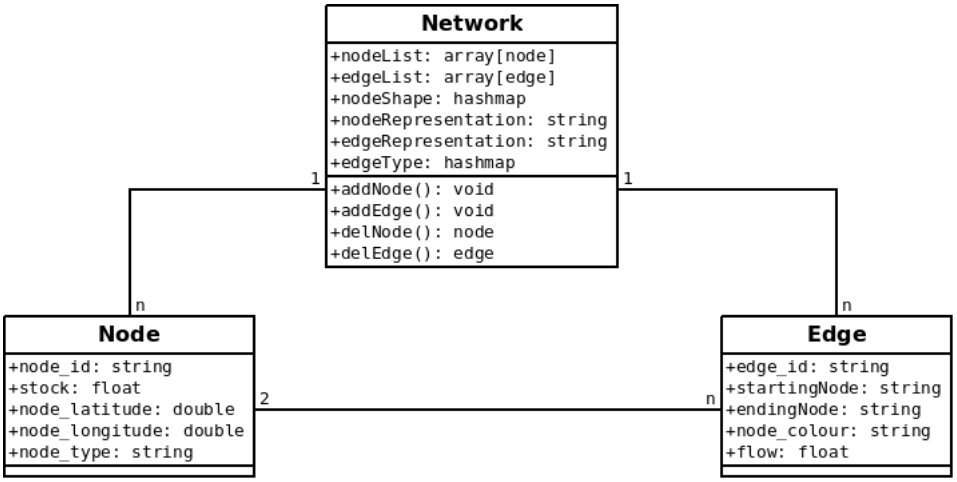

Figure 3: UML Class Diagram

3.6 Data Flow Diagram(DFD)

- Level 0

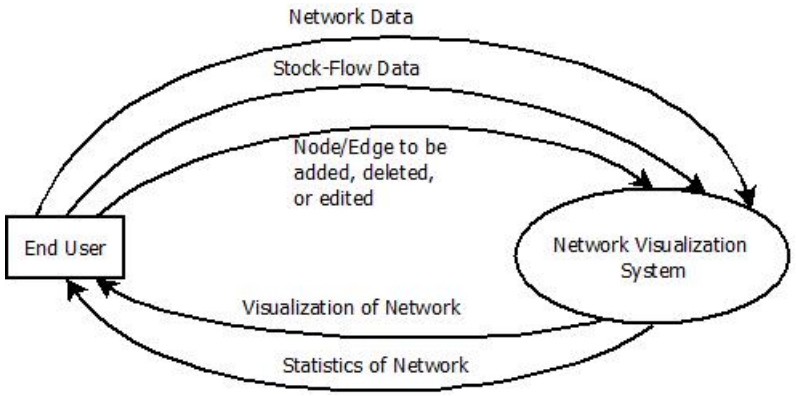

Figure 4: Level 0 DFD

- Level 1

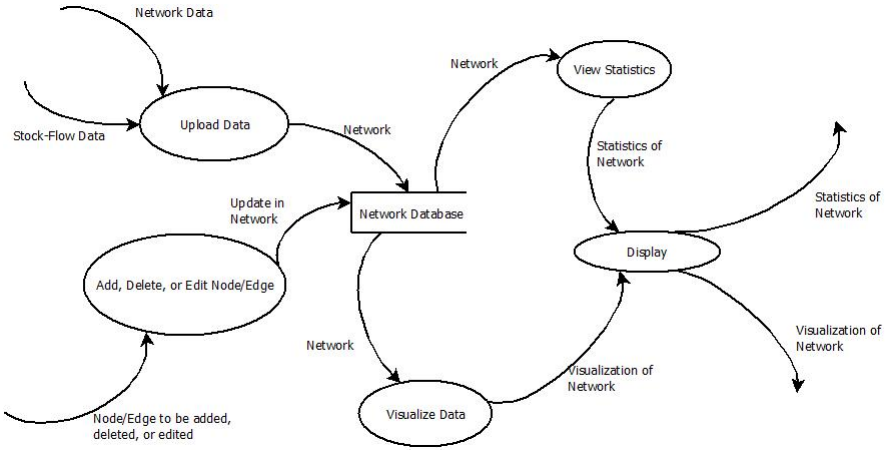

Figure 5: Level 1 DFD

4 User Interface Design

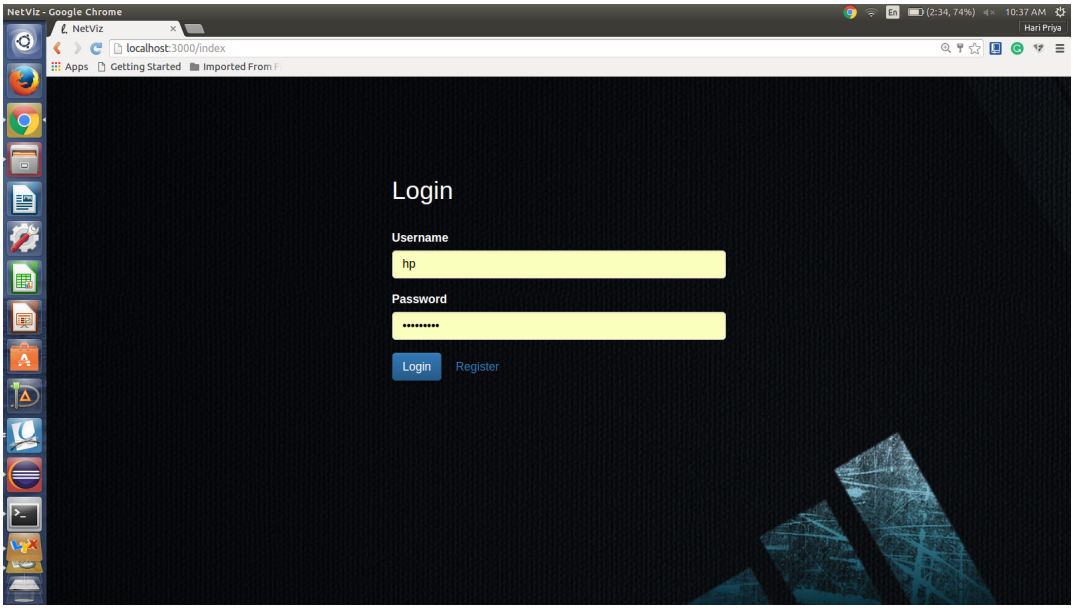

Figure 6: Login

This is the *'home'* screen for our application. The user has two options : *'Login'* for an already existing user and *'Signup'* for a new user. The existing user enters

the user name and password which is verified with the database. When a new user clicks on the *'Signup'* button, he/she is directed to *'Register'* screen. In case the user enters invalid user_id or password then a small error message would pop up requesting the user to enter his credentials once again.

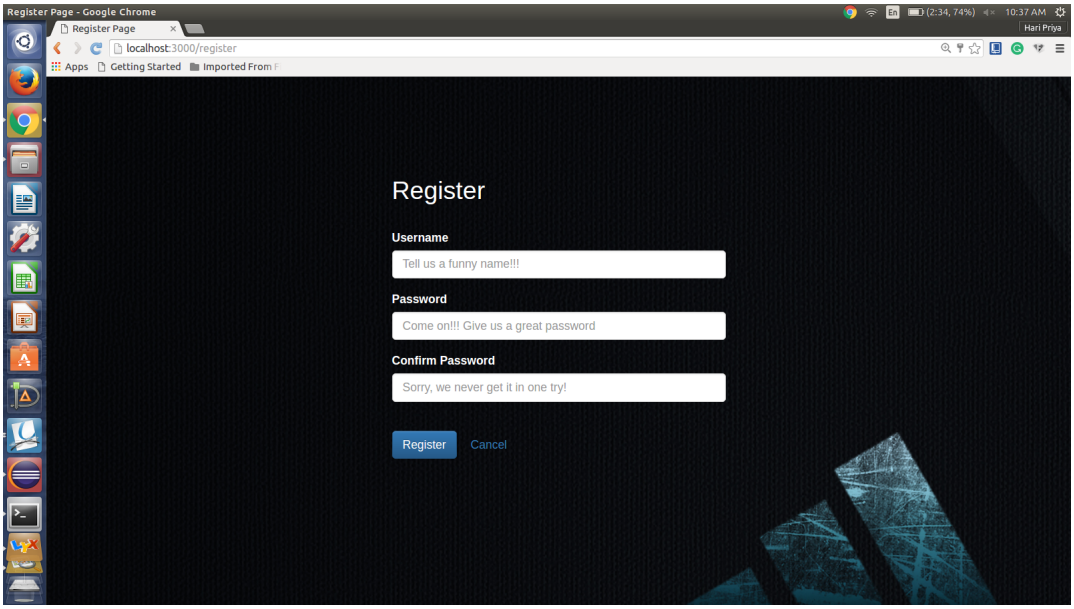

Figure 7: Register

A new user is directed to this *Register* screen on clicking the *'Signup'* button on the *'home'* screen. The user enters a user name and a valid password which is at least 8 characters long and has at least one numeric character.

There are four tabs : Nodes, Edges, Visualization and Help. An additional option is provided to the user if they want to directly upload data from csv. He could also manually add the data for nodes and edges separately.

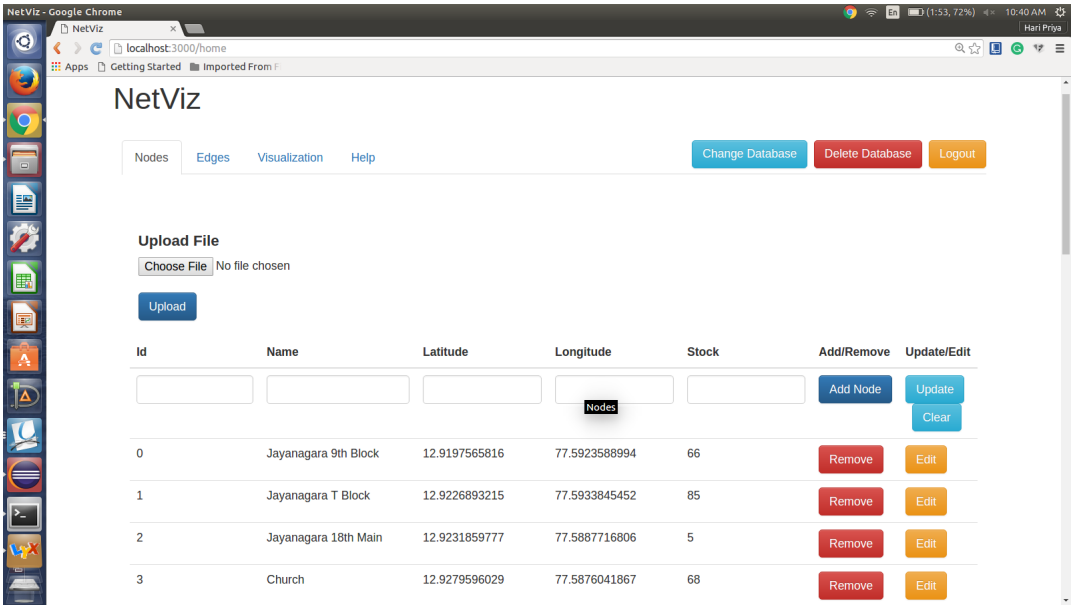

Figure 8: Nodes

This screen shows the nodes and its attributes. Each node has an unique id, latitude, longitude and stock data. The node data is taken as a csv file.

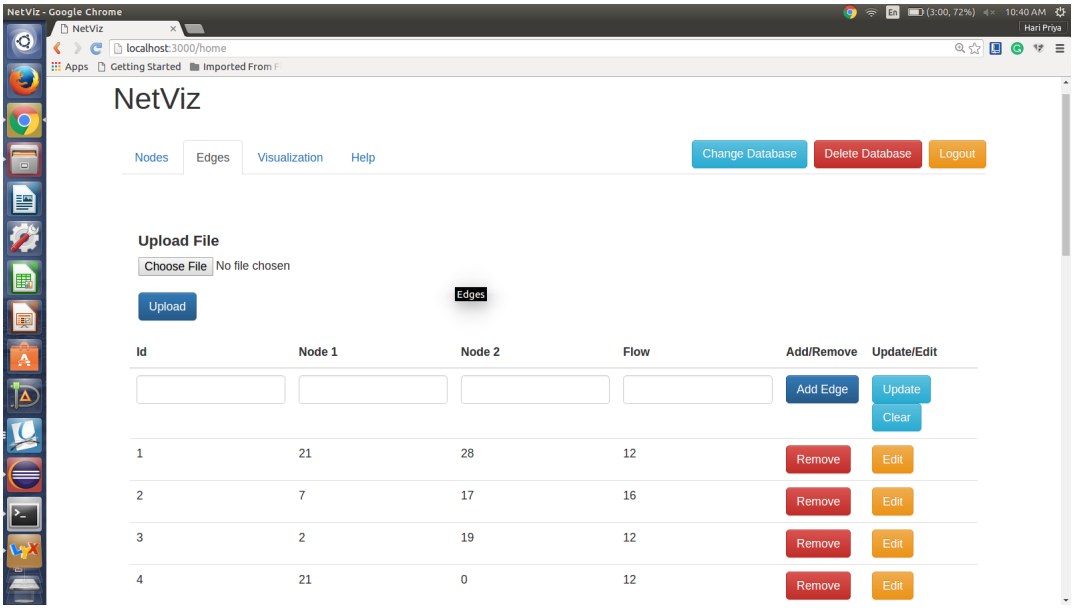

Figure 9: Edges

This screen shows the edges and its attributes. Each edge has an unique id,

starting node id, ending node id and flow data. The data related to edge is taken as a csv file or can be added manually by the user.

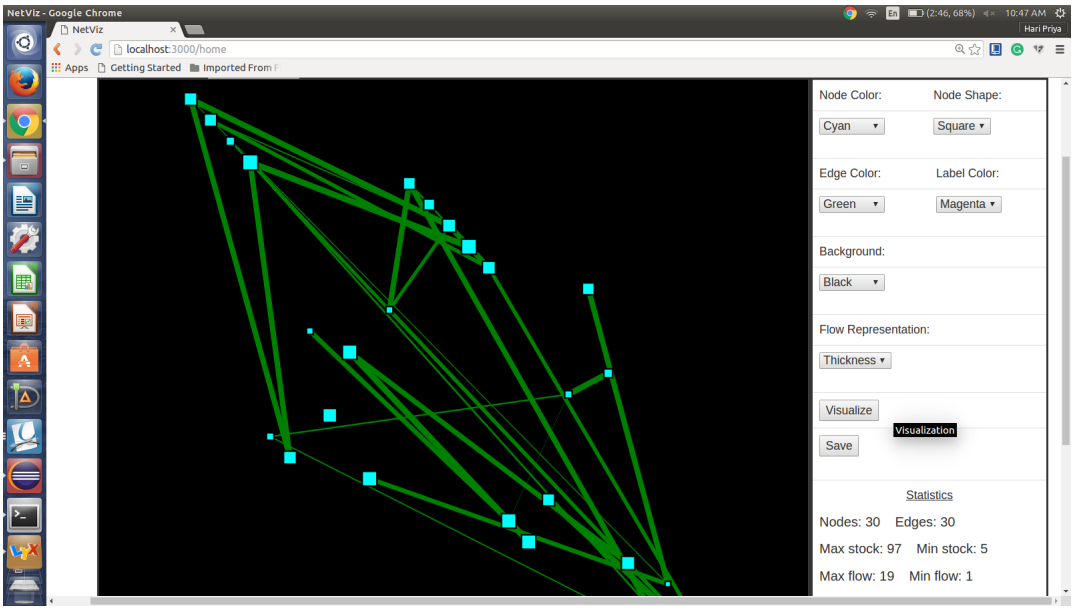

Figure 10: Visualization - 1

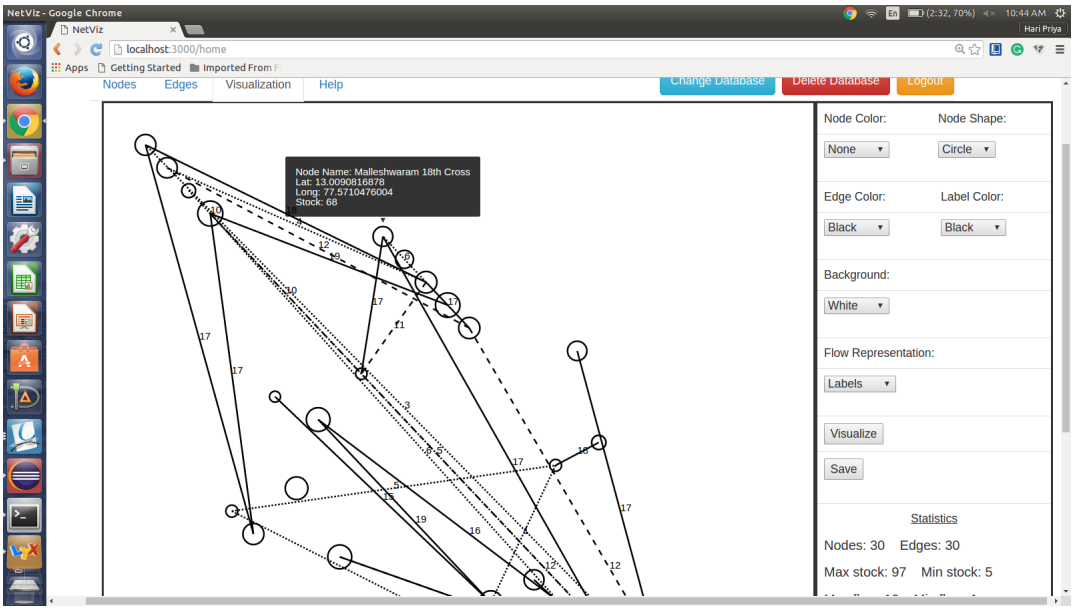

Figure 11: Visualization - 2

The *'Visualization'* tab shows the visualization of the stock and flow data. Some statistics are shown on the left side of the screen. For example : the number of nodes and edges. The user can change the visualization by changing how the nodes and edges are represented. For example : the user can change the shape of the nodes. In case the user visits the visualization tab without entering any data. It would display a text message "No network to visualize".

User can also change the stock-flow representation. For example : the user can choose to view the flow data explicitly or can have a relative estimate by the thickness of the edge. The stock data can be visualized by the size of the nodes.

4.1 Usecases

4.1.1 Usecase Diagram

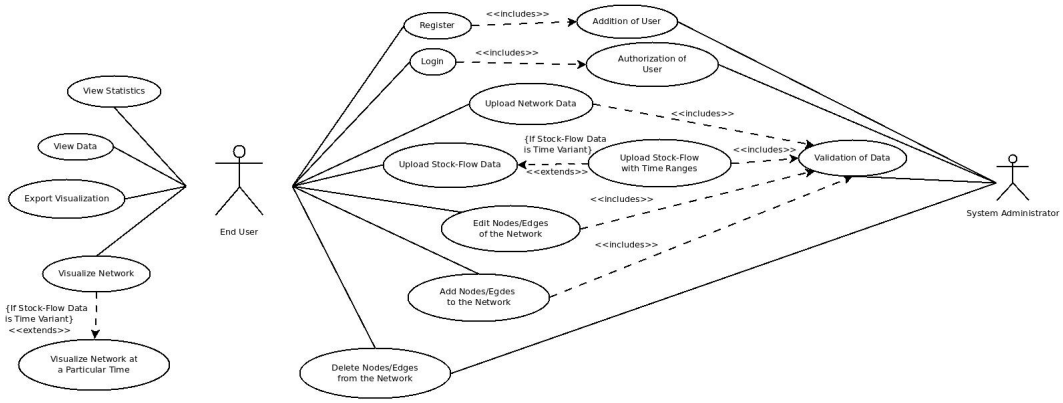

Figure 12: Usecase Diagram

4.1.2 Sequence Diagram

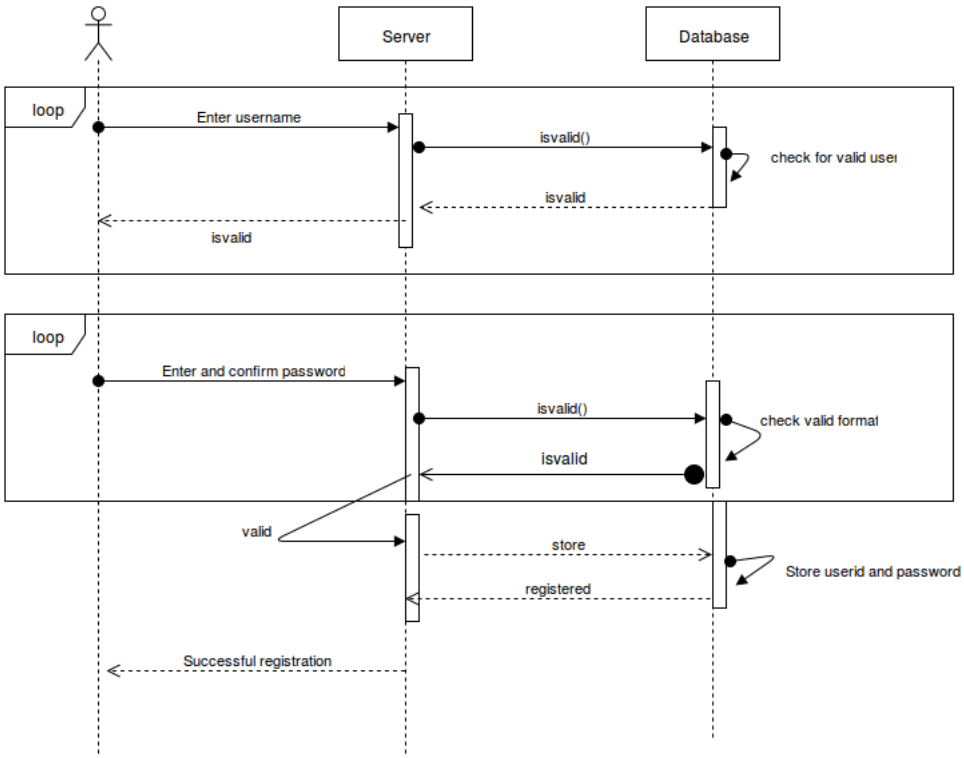

Figure 13: Registration

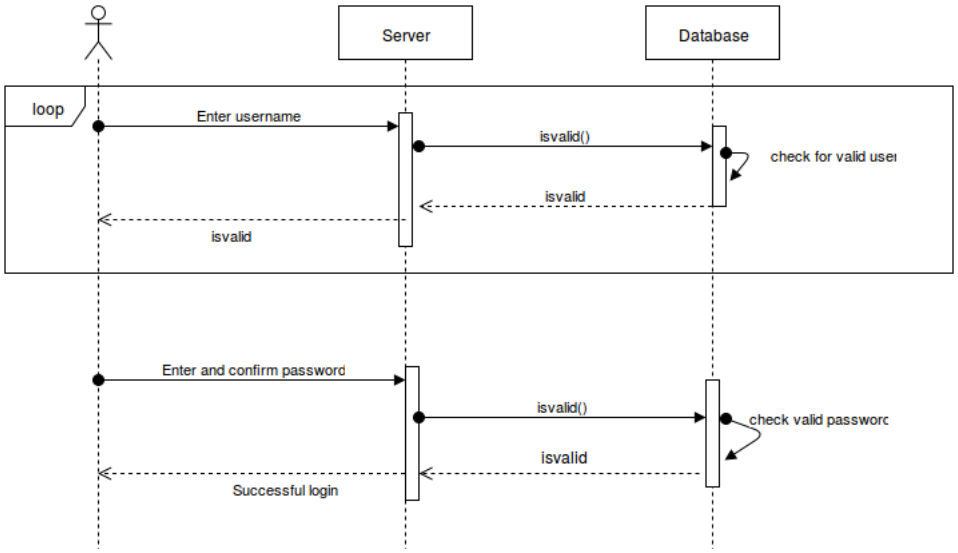

Figure 14: Login

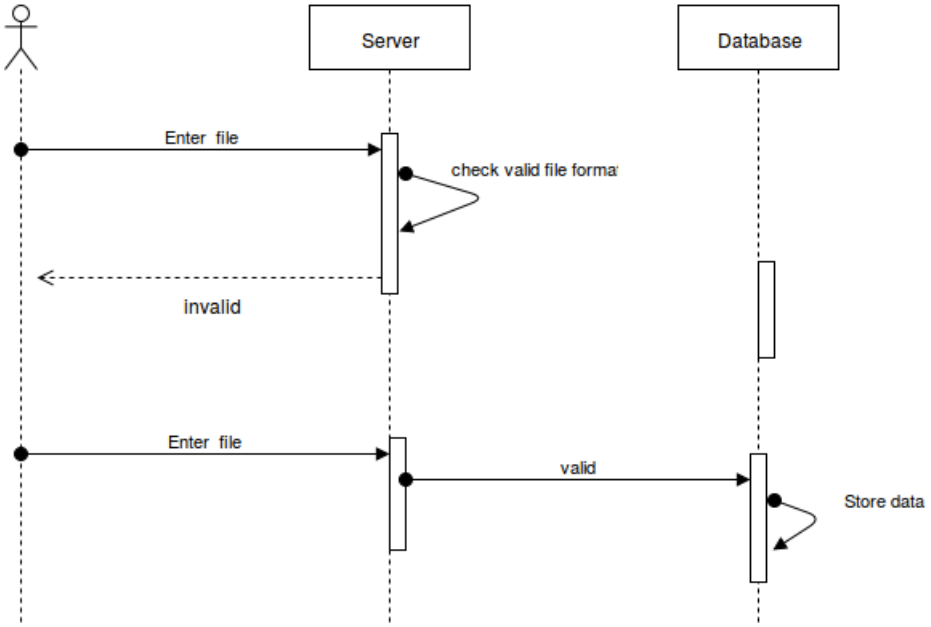

Figure 15: Data Import

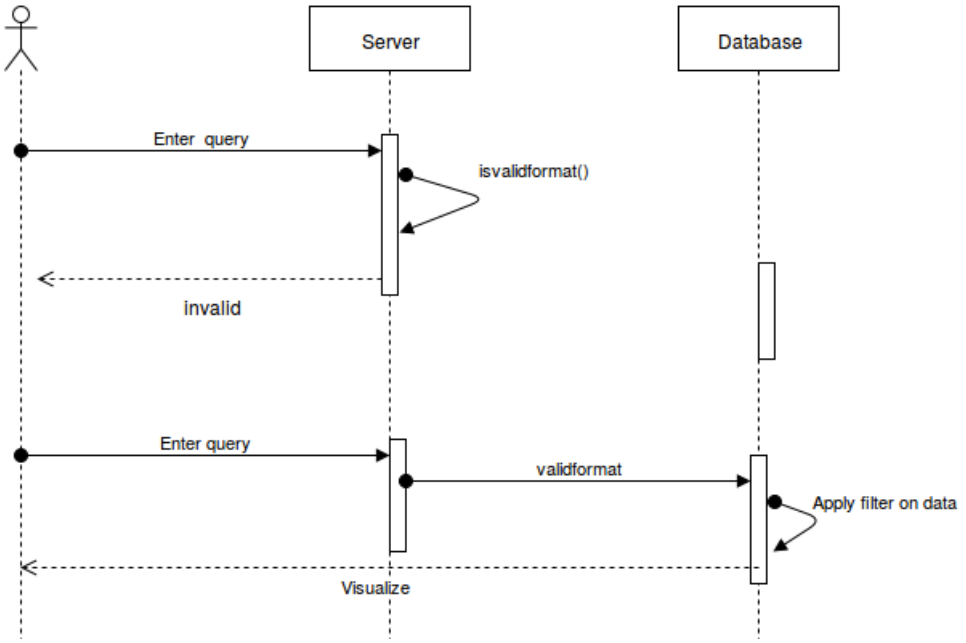

Figure 16: Visualize

5 Restrictions, Limitations and Constraints

- The data files must be in csv format with the columns for each file as specified in section1.2.
- We assume that any prepossessing needed to bring the data in specified format is done by the user beforehand.
- If the server is not on the same local machine as client, the client must have internet connection.
- The visualization of the data must be done quickly and efficiently. It should not take more than 1-2 seconds to visualize even large networks.
- The visualization rendered has to be topologically correct even for large networks.
- Availability is not much of a problem for this application. So, even if the program is not available for a few moments, the user is not affected in any major way.

- The application must be built in a way such that any new additions can be made without changing the existing functionality of the program.