Introduction

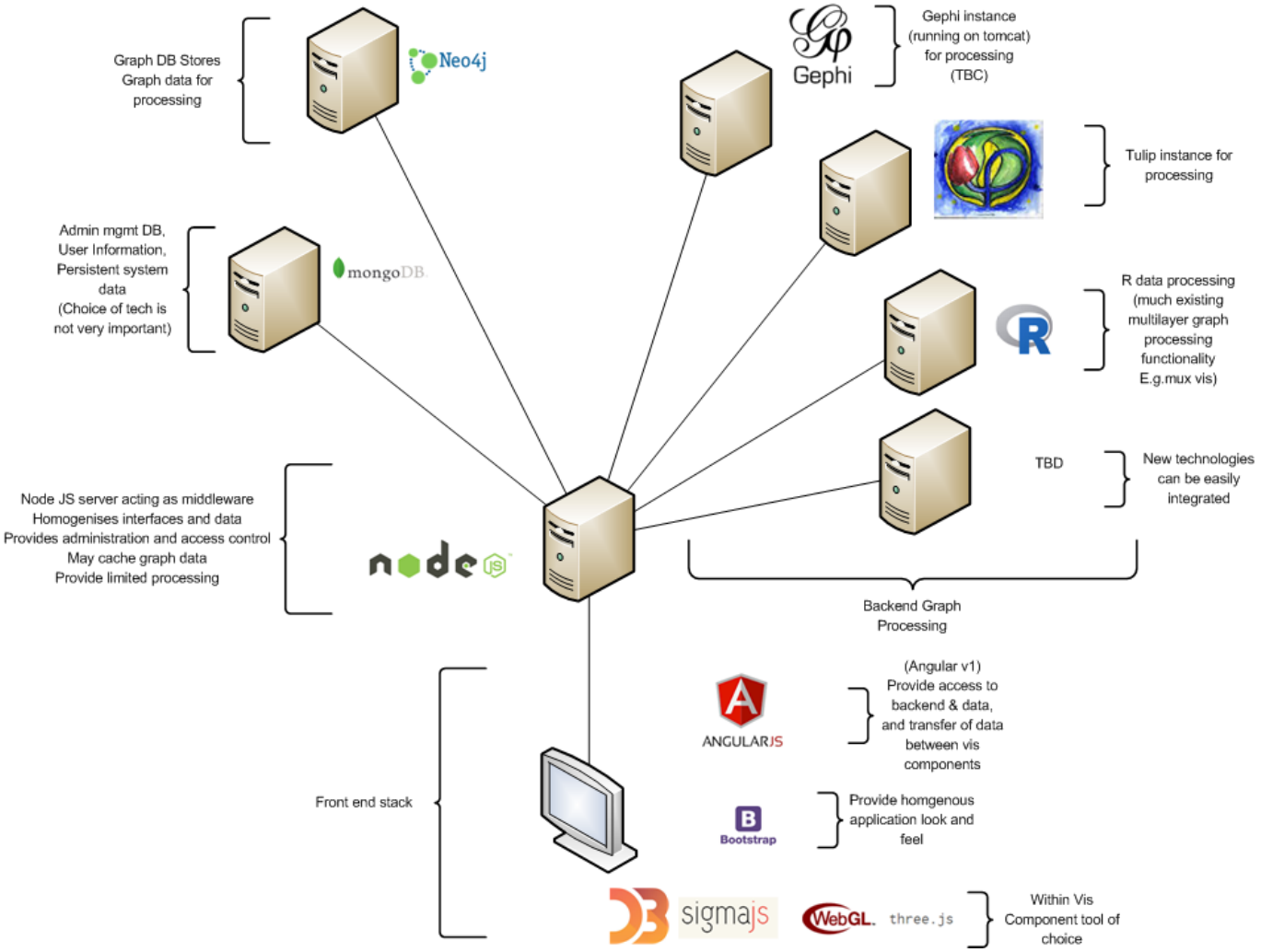
This document describes both the architecture for the front end (web based), middleware and back end components of BLIZAAR project platform, as well as describing how to setup and install all components of the platform. The goal of this document is to ensure all project participant understand full how the system is built and how each component interacts, and how to install the BLIZAAR platform .

Terminology

As this is a graph visualization based project there are many different terms to describe the same entities. Within this documentation we will attempt to be consistent , however as the framework includes many different subsytems and APIs , it is not always possible. When referring to the entities in a graph we refer to them as “nodes”. While often this term is only used to refer to them when visualized, we use the term in all contexts. Other systems frequently use the term vertex, in place of node, and vertices in place of nodes. For all of our purposes within this documentation these terms are equivalent. When referring to the relationships between nodes we use the term “Links”. Again this term originally refers to the visual aspect . However many libraries we use frequently use that term rather than the more visualization neutral “edges”. Again the term edges can be found in multiple different systems, however in the context of this documentation the terms can be considered equivalent Depending on their background of a person they may be more used to referring to a set of nodes related by links as a “graph” or a “network”. We will use the term graph, however in the context of this documentation the terms can be considered equivalent

System Architecture

The system is centered around a node.js middle-ware web-server and Neo4J back end. For details about the versions of each piece of software see the BLIZAAR toolchain setup section. See figure for the full system architecture.



Back End Components

Nodejs Server

This server acts as middleware for the entire project . It serves all webpages to the front end , and is also responsible for user access rights All requests from the front end pass through this server to their target back end components whether it is for data retrieval from the graph database or for processing of the graph data. The Nodejs middleware stores an instance of user's current graph for each currently logged in user, which can be passed to the various backend engines for processing and updated form the neo4j database The Node JS sever contains several modules , each of which is a separate file and provides different functionality.

app.js.

The app.js file provides the interface to the server. All rest API calls are received here and passed to the relevant module.

system.js

This module interfaces with the MongoDB, and is responsible for user access control. All API calls are authenticated via this model (as the MongoDB stores user info). As the MongoDB is also used for serializing (storing) user graph data between sessions, as well as caching data (such as the node and edge types for each graph type to speed up queries), that functionality is also stored in this module.

graph.js

This module stores and process the graph on the server side. It is updated via queries to the neo4j database, and graphs loaded from the mongoDB vis system.js.

graphDB.js

This module is responsible for interacting with and querying the neo4j database. It takes requests via app.js and transforms them into cypher queries..

rServer.js

This module handles requests concerning graph process and passes them to and rServe instance. It transforms requests into a suitable format and updates the graph with the returned data. It also queries the rServer for available processing functions, e.g. clustering and layout.

Neo 4j server

The neo 4j server stores all project master data sources. It is accessed via its built in REST API from the node.js middleware server. All requests for data from the front end or other components should be made to the middle ware which then queries the neo4j database. The various different data sets are distinguished in the database using neo4j node labels.

R Server

The back end R server is used to process graph data. R scripts are stored within in the R engine are remotely invoked from the nodejs server’s R server component. Acces to a running R instance is provided by Rserve, the standard R server application available with all R installations.

Mongo DB

This data base store information related to user profiles and access rights. We have chosen this DB rather than storing this information in the graph DB as it integrates easily into the neo4j software stack with minimal overhead, and user information does not require the use of a graphDB. Additionally, we aim to keep a clean common master data DB that can be shared across all users. The

mongoDB is also used as a cache for graph data. Graphs can be saved by the user and these are stored in the MongoDB for quick retrieval . The various node and edge types for each graph are also cached here, to allow for quick access.

Tulip

This component provides additional graph processing functionality and is to be accessibile by the middleware. Currently this component is not integrated into the system. Gephi / other Modules Other components such as Gephi (running on a tomcat server) can be integrated to provide functionality not offered by other graph processing components

Front End components

Angular.js

Angular.js is the framework being used for front end development. It offers a robust, proven frame work for website design. For more information see section

Bootstrap

Bootstap is a css library that offers a consistent look and feel to all front end webpages. It is frequently used in angular projects

D3.js/ WebGl / Visualisation development

Development of front end visualizations is not constrained to any specific technology. Project researchers are free to use any available trools, such as d3.js, webgl sigma.js or any technology of their choice. Currently d3.js , sigmal and webgl have been tested and shown to not have any significant issues interacting with the rest of the framework.

Graph Structure and Storage

Graph Data Structure

We use a very basic graph data structure modelled on the common types of structure used by d3.js when processing graphs. At its most basic, a graph object consists of a simple JavaScript object with two properties , nodes and links. Each link contains an id of its source and target rather than a reference to the node object . This is to allow easier transmission of the graphs via json, as a reference cannon be properly encoded in a json message without duplicating objects. To allow fast

lookups of nodes and links, a look up table (simply a JavaScript object used as a property map) for each is calculated in the middleware . For nodes it is called node, and the key is the node id. For links it is called link and the key is the link Id.

Master Graph Data

As part of this project all primary input data sets (e.g. the histograph data set, protein interaction data sets, metabolite interaction data sets etc.) for both application domains are stored in the neo4J back end graph data base. This master graph data is the source form which users build their own graphs. Neo4J uses the field “id” to identify nodes uniquely. We also use this a unique identifier for all of our nodes throughout the framework.

User Graph Data

A user builds a graph by making queries from the front end to the back end master graph data sets via the middle ware. To avoid having the front end having to pass the full graph to the middle ware for every query we store a copy of each users current graph on the server. As well as reducing load between the front end and back end, this simplifies saving graphs and work in progress on a per user basis

Neo4J Graph DB structure and Terminology

We store all master graph data in Neo4j. Regardless of the application domain, all input graph data is stored there. Within neo 4j node labels are used to identify sets of nodes. A node can have multiple labels. Each back end graph is distinguished by a different label. Additionally labels are used to distinguish different types of nodes. For example within the histograph data sets, all nodes have a label “histograph” and nodes describing people have a label “person”, as well as “histograph”. Histograph node describing places will have a label “place” as well as “histograph”. Edges also have can have a type specified , which can be used to restrict the edges which come back associated to nodes in a query. Edges and labels querying has specific semantics in (the Neo4j query language), however all of this should be invisible at the front end. We merely describe the label convention here to help understand better how queries for graph data are formulated, passed from the front end to the middle-ware, translated into cypher in the middle-ware and passed as a query to the back end.

BLIZAAR Tool chain setup

Application Framework Tools

The following components / tools are necessary to set up you BLIZAAR project so that the applications will run. The version of each that has been tested is specified, using other versions that contain a major release may cause issues, however minor version differences should generally be fine. For

windows users installers for Neo4j, MongoDB and Node.js are available from the project git hub and can be downloaded from the repository http://blizaar.list.lu:5001/mcgee/blizaar_windows_tools

GIT 2.9.0

Git is out version control tool. It is used to retrieve the project code and data and integrate changes from all team members in a single repository. When you download GIT and install git (on windows) , another tool called git BASH is installed. This is a command line shell that allows for access to git commands, and is also useful for writing scripts and executing scripts It is available from https://git-scm.com/

Neo4J (3.0.3)

Neo 4J is our graph data base and we will be using it to store our graph data. As it is a graph database it does not use SQL, but rather its own query language Cypher. It can be downloaded from here. We are using the Community edition. It is best to download the ZIP version, as this allows neo4j to be started via a script. Neo 4j has a dependency on java . If you have issues running it download the java jdk at the following link, an ensure that the JAVA_HOME environment variable points at its root folder. Download the 64 bit version (Windows x64 or luinixX64) from the following address http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Mongo DB (3.2.7)

Mongo DB is a SQL free database that we are uses to store system administration setting and information. We also will be using it to cache user data and preferences. It can be downloaded from here https://www.mongodb.com/download-center?jmp=nav#community

Node.js (6.5.0)

Node is our application server . It will be our webserver and middleware route all messages form front end to back end It can be downloaded from here

R (3.3.x)

We use R for some back end computations, via the R serve package, providing the interface for the node.js server. Setting up R for the project requires some additional action described later.

Rstudio

Rstudio is an environment for working with R and is much easier to use than the default interface.

Google Chrome

As this is a web tool, a browser is necessary. Many people already use chrome and it has been shown to work well with existing visualizations. Firefox is acceptable too, however internet explorer will most likely not work.

Javascript Development libraries

There are many many development libraries available for javascript. All javascript development libraries, should be checked in as part og the GIT repository so all developers are accessing the same version.

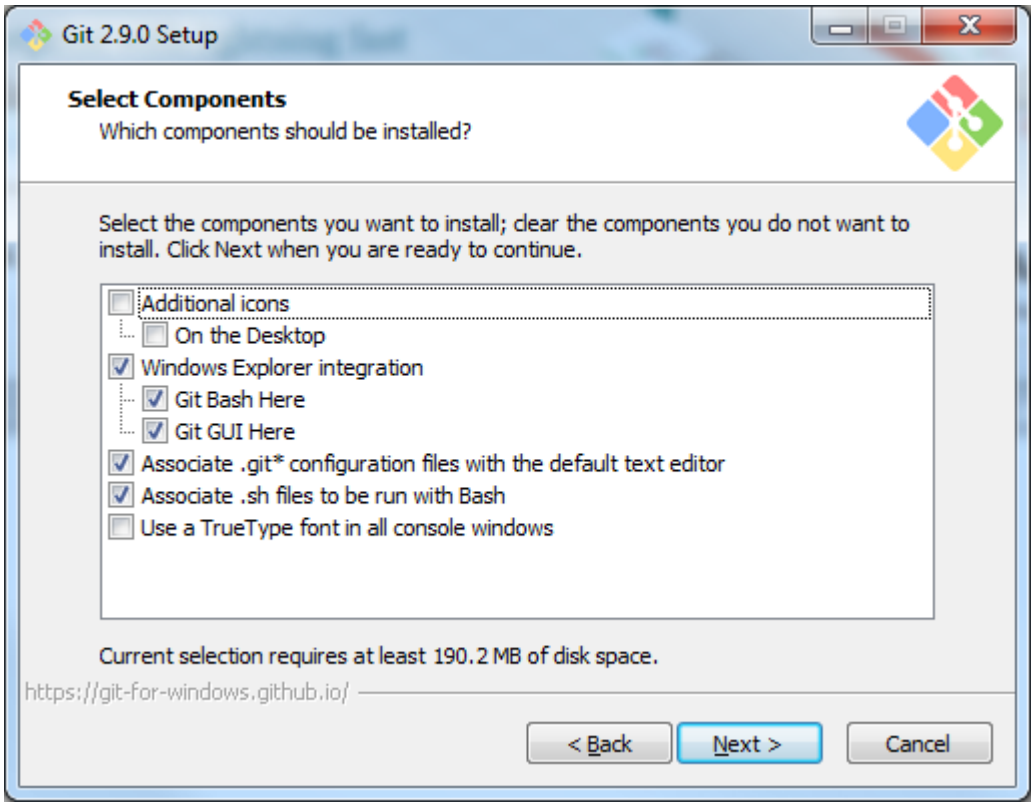
Installations Prerequisites:

Download each of the preceding applications, and ensure that you have access to the project repository with your GIT username and password

Setup Procedure

1. Install Git

Git also installs git bash which is a bash based shell which is very useful for not only using GIT from the command line, but also for scripting. Integrating the command “Git BASH here” unto the right click menus (and option available during the install, see image) is a useful feature. It is best to only use git BASH for command line control of GIT (so there is no need to enable it for use form the windows command prompt)



Once git has been installed , it is worth setting up an ssh key to simplifying using git and checking out of data. An ssh key allows a user on a specific machine to use git operations without having to enter a username and password every time. See the section for more details.

2. Clone the repository

Get the project files as follows: Open a git bash window and navigate to the directory you would like to store the project in and type in the following:

```
git clone --recursive https://git.list.lu/BLIZAAR/blizaar_neo4j_DB.git
```

3. Install Neo4j

Extract the zip tool to your chosen neo4j directory and note the path. Edit the *neo4j_start.bat* and *neo4j_stop.bat* files in the project's root directory, to point at at your installation.

4. Install MongoDB

Install MongoDb using the downlaoded installer and edit the *mongo_start.bat* file in the project's root directory, to point at at your installation.

5. Install Neo4J

Use the Neo4J installer to install Neo4J.

6. Clone the master Neo4J backend graph DB

Clone the Blizaar_neo4j_DB project and copy the blizaar.graphdb subfolder of the blizaar_data subfolder of the project into the databases subdirectory of your Neo4J installations “data” folder.

If you have an ssh key set up clone it with:

```
git clone git@git.list.lu:BLIZAAR/blizaar_neo4j_DB.git
```

If you have no ssh key set up, you can colen it over http, however this is slower and less reliable:

```
git clone https://git.list.lu/BLIZAAR/blizaar_neo4j_DB.git
```

Another option is to download and extract the zip file from the LIST Gitlab instance installation at: https://git.list.lu/BLIZAAR/blizaar_neo4j_DB

7. Configure Neo4J

Edit the Neo4J config file (in the /conf/neo4j.conf subfolder of your Neo4J installation) to point at blizaar.graphdb. i.e. set the following parameter:

```
dbms.active_database=blizaar.graphdb
```

8. Install R & RStudio

Install R and R studio, and then open the R.rproj file int he R subfolder of the BLIZAAR platform installation directory. Run the script firstTimeSetup.r with the following command.

```
source('./firstTimeSetup.r')
```

8. Install Required Node Packages

The node.js middle-ware requires packages to be installed, fortunate node.js provide a package manager. To install all required packages run the following command at the command prompt in the BLIZAAR installation directory (you can use the regual windows command prompt, to git bash..

```
npm install
```

Running the Platform

To run the latform each of the components needs to be started: Neo4J, Rserver, MongoDB, and the

node.js middleware.

1. Start Neo4J

Start Neo4J by running the *start_neo4J.bat* batch file in the platform home directory. This file MUST be run as **admistrator**.

2. Start MongoDB

Start MongoDB by running the *start_neo4J.bat* batch file in the platform home directory. This file must **NOT** be run as admistrator.

3. Start RServe

Start RStudio by opening the *R.rproj* file in the R subfolder of the platform home directory. Run the following command to start RServe:

```
source('./startRserve.R')
```

4. Start the node.js server

Open a command prompt and navigate to the BLIZAAR platform home directory. Type the following to start the server

```
node app.js
```

The first time the server is run it will automatically create the MongoDB database file with a default user and username.

5. Login to the application

Open the chrome browser and enter *localhost:3333* in the navigation bar. The default username is blizaar and the password is blizaar.

SSH keys

In order to simplifying checking in and checking out of data (and not having to enter a password every time) generate an ssh key pair and add the public key to gitlab in your gitlab profile settings, under ssh keys. This is strongly recommended as it will allow you to clone etc. via ssh. For example:

```
git@git.list.lu:BLIZAAR/blizaar_platform.git
```

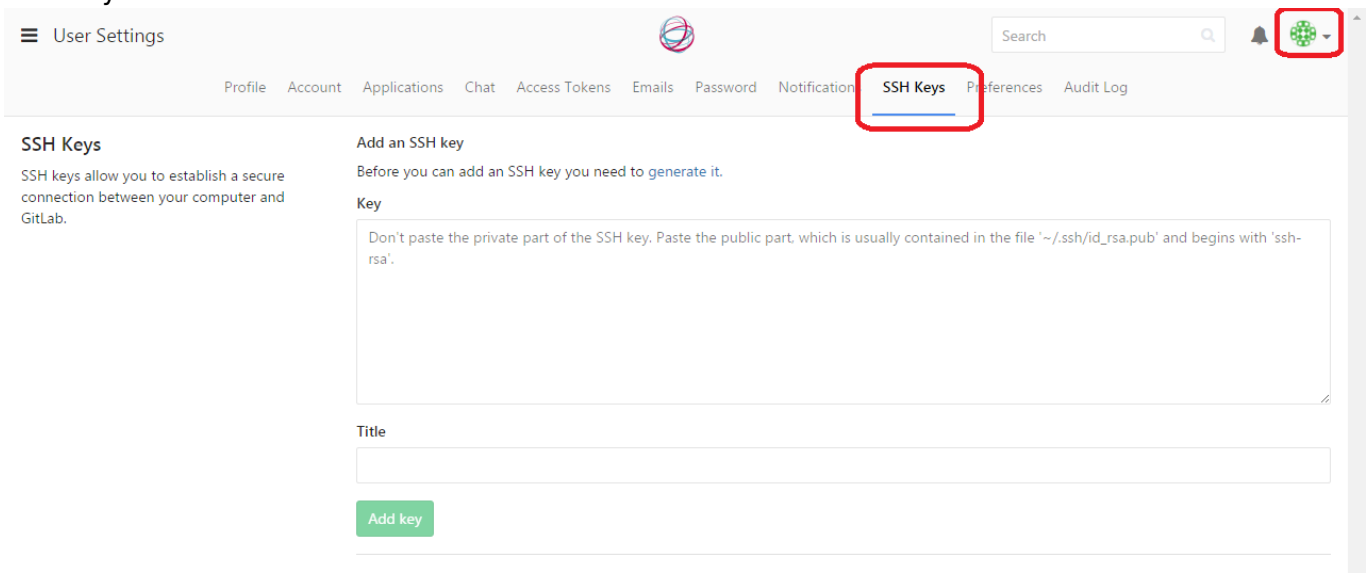
Using ssh is quicker, more secure and more reliable than using http, in addition to not requiring a username and password for every operation. To generate your ssh key open up a git bash window and type the following

```
ssh-keygen
```

Follow all onscreen instructions and once the generation is complete you public should be generated. It can be found in a sub-directory of your systems home folder (this varies from system to system) called *.ssh*. The file will have the extension ".pub" and the contents will look similar to the following:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABIAqzQQEAvyF0awxZMohnsdfnLWmg7Yd5tXvQqMEmpVi6jvAWLNrI2/7+
VMitAg1gOEJGHJolf/wiGLRMsSMXvFgHNPoAPnlApePiqsLnLSueCyGXZfKp+QUxEFUpblMM8E8p
Ti9BnWlrb21ZrzOZXcC0GSpo+h4zgt78Bjj7n79PboULCiXZopcTH7n6eT/Abyp1hXf/9eqgomL0
2DWPi5xf55laHBPc+BSusVa7M52bpWQA1ET80gamCqSaXTYkVBsq+uFqY4qvpbKJiMLm+S3SOHzI
mgmPVxlhF1YOhICDh8gE6w+D1LAMTWTdwdCu9Ayh3lbSuM/mZVaMxkekEeO7U/dj4w==
yourname@computername
```

To add your to the git lab server, login (https://git.list.lu/BLIZAAR) and under "profile settings" (click on you profile icon in the top right corner and select settings) and once the pages loads select the "ssh keys" tab



Paste your public key into the space provided and then click on "Add Key"

More detailed instructions to generate an ssh key pair (using git bash) can be found here. https://git-scm.com/book/en/v2/Git-on-thrver-Generating-Your-SSH-Public-Key

GIT GUI Integration

Torotise GIT is a useful too integrates GIT functionality into windows right-click menus, and makes for simpler use of GIT for those who prefer to not have to use the command line. it can be downloaded from https://tortoisegit.org/download/