Architecture Design Document

1 Introduction

This document describes the architecture of our software product TAGC genome visualizer. First it will state the design goals, after which the software architecture views will be described. There is a glossary available at the end of the document.

1.1 Design goals

Scalability

The genome graphs or phylogenetic trees loaded into the program won't slow it down with increasing size. The system will only be limited by the amount of hard disk space available.

Reliability

There will be no user interaction that can cause the program to crash. Once started the system will run flawlessly until the user explicitly closes it.

Security

There will be no vulnerability in the program that allows others to steal the data loaded into it by the user. The data supplied and produced by the user will stored securely in a database.

Performance

The program will respond to user input without noticeable delay. It will use an architecture and algorithms that will optimize the amount of time needed for the program to process the user's data.

Compatibility

The program will be compatible with the *.gfa* and *.nwk* formats for storing genome graph and phylogenetic tree data respectively. When supplied files in these formats the program will be able to visualize and process the data in accordance our client's requirements. Furthermore the program will be able to run on any system that has the Java runtime, and either Google Chrome, IE or Mozilla Firefox installed.

Usability

The program will have a user interface that is intuitive and easy to learn. It will also support all the functionality that has been specified as part of our client's requirements.

Maintainability

The architecture of our program will be designed with future updates in mind. It will be possible to update the program remotely over an internet connection in a safe and efficient way.

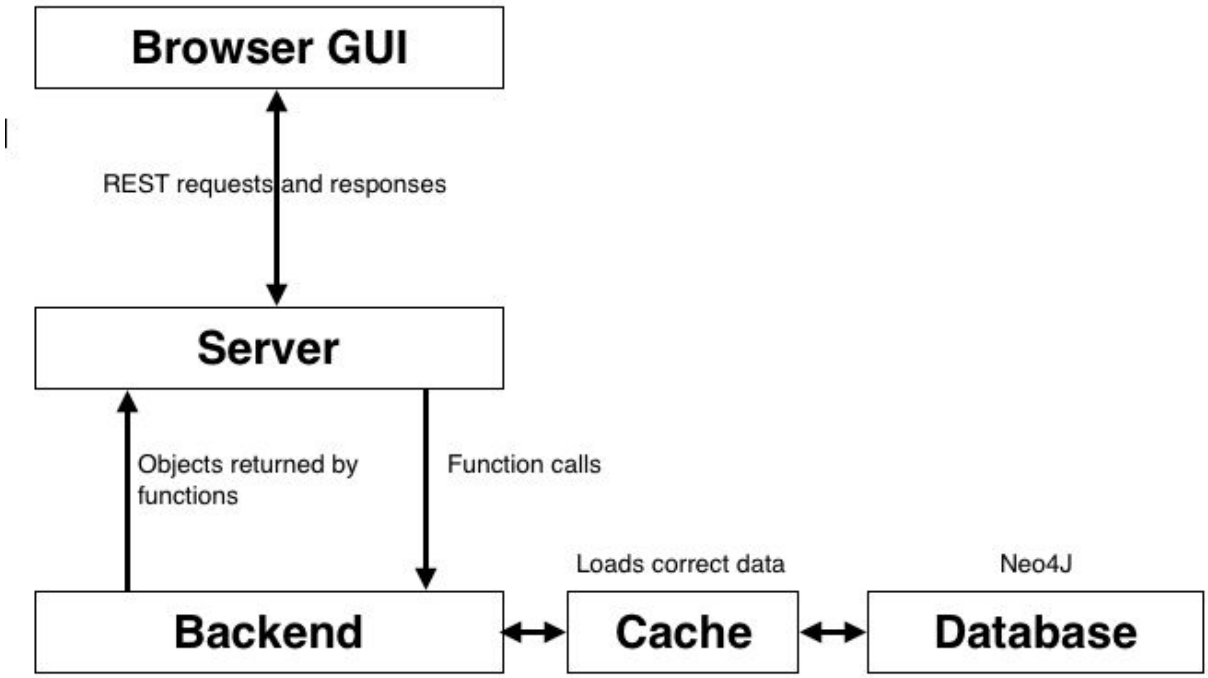
2 Software architecture views

2.1 Program structure

Our program is build in three different modules. The first module is the module that the user will see, the browser GUI. This is written in JavaScript and is located in the resource folder of the TAGC webapp. The two models that the user doesn't directly interact with are the restful API and the backend, both of which are written in Java. The restful API is also located in the tagcwebapp folder, and the backend is located in the backend folder. The backend can further be split up in the Java classes which keep their data in memory, and the database which stores the complete data. A caching mechanism will be implemented to minimize the data flows from the database, which will eventually also include some form of lazy loading to improve the user experience.

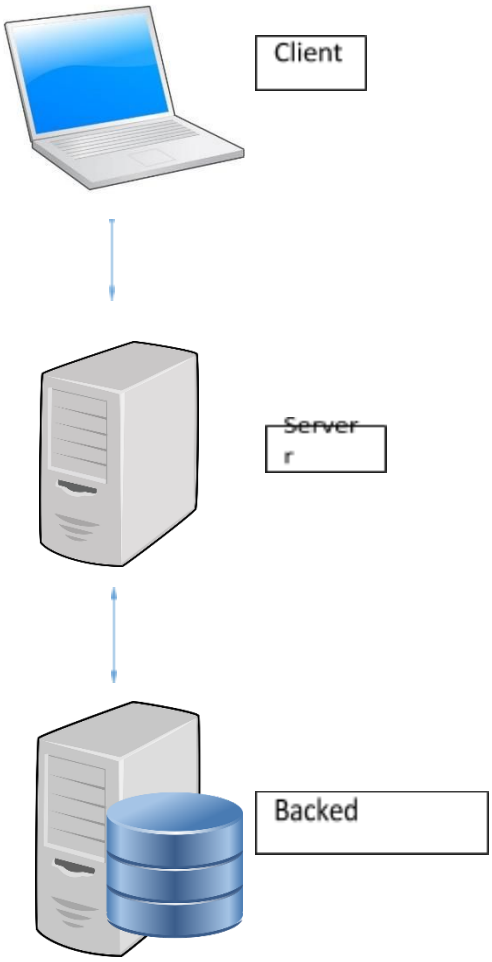
2.2 Subsystem decomposition

The user interacts with our program through a GUI displayed in his browser. Data gets exchanged between the browser and the server through a restful API. When a user causes the browser to make a REST request to be sent to the server the server then calls the appropriate functions in the backend module. This module then processes the request directly, using a cache mechanism between the data in memory and the database. If the function has a return value it returns this to the server module. The server module then sends it in a response to the request from the browser. The interaction between the server and the backend module is implemented using the adapter software design pattern. This allows the interface used by the server and the interface of the backend to be independent of each other.



2.3 Hardware/software mapping

In its simplest form the program can run as a whole on just one machine that has a screen and mouse and keyboard to interact with. However it will be possible to use the GUI through a browser on a different machine from the rest of the system and interact with it over a network or the internet. It will also be possible to connect to the database if it's located on a different machine from where the program is running, although this is mainly reserved for future use and is not necessary at the moment.



2.4 Persistent data management

The program accepts the *.nwk* and the *.gfa* file formats. It parses these files and stores them in a database (by creating CSV's which are read by the database) on first use, after which the database will be used for loading the correct data whenever a user interacts with the program. Data is stored and loaded in the database in such way to minimize the amount of memory needed during program execution, using a cache to minimize the relatively expensive disk reads. The database itself is in Neo4J, a relatively novel graph database. This allows us access to relatively complicated features such as indexing, while minimizing the amount of code that needs to be written for this. The database can be queried using Cypher, a graph based equivalent of SQL.

2.5 Concurrency

Concurrency at the webserver is handled by the Grizzly NIO framework that the server is implemented in. This allows for multiple users to simultaneously use the program from different client machines. Concurrent data usage is going to be achieved by the use of a DBMS.

Glossary

Genome

A genome is the complete set of DNA of an organism.

Phylogenetic tree

A tree that shows the evolutionary relationship between organisms related through their DNA.

DBMS

Acronym for database management system. A system which creates and manages databases in a secure and efficient way.

GUI

Graphical user interface. The program component responsible for what the user gets to see and user interaction.