Develop an RDF Store for Phenotype Data.

Final Report for CS39440 Major Project

Author: Adam Connah (aoc9@aber.ac.uk)

Supervisor: Georgios Gkoutos (geg18@aber.ac.uk)

24th April 2013

Version 1.0 (Release)

This report is submitted as partial fulfilment of a BSc degree in Internet Computing (H621)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality

Consent to share this work

Acknowledgements

First and foremost I am grateful to my project supervisor, Georgios Gkoutos, for the guidance he has given me throughout the project. He has shown great faith in my abilities as well as a passion for the project which has inspired me to produce the work to the best of my ability. Additionally I would also like to thank Robert Hoehndorf for selflessly lending his technical expertise to the project, without which the project would not have been as successful as it has been. Neil Taylor is also thanked for encouraging me to push my abilities and take on this project to begin with, along with all the Staff at Aberystwyth University during my tenure here that have helped to provide a great learning environment without whom I would never have got to this position.

The support from my friends both at home and university has been invaluable and I will take this opportunity to thank Ben Shone, Michael Chajnus, Adam Rigby, Sebastian Graham and Rhys Palmer who have supported me greatly in life and through my studies. A special mention also to Charlotte Lewis who has always provided me with support and an escape from any troubles that I have encountered

Finally, I am forever indebted to my parents their understanding, endless patience and encouragement, without this I could have never dreamed of becoming the person I am today.

Abstract

Since the 'post-genomic era' (Robert Hoehndorf, Paul N. Schofield and Georgios V. Gkoutos, 2010) there has been a rapid increase of quantity and quality of biological data. This increase in data has led to the development of large numbers of databases for the purpose of storing, retrieving, analysing and integrating them together to gain new knowledge. The biomedical research community has invested a considerable amount of resources into creating ontologies which define the terminology and relationship between vocabularies. In order to maximise the potential of these vocabularies the semantic web must be exploited to its full capabilities to provide access to the data as well as building knowledge about the data. However,, Biologists lack the necessary skills to do this, and so this is where Bioinformatics bridges the gap between the two domains.

The project has been able to provide this 'bridge' for Biologists, by creating a web application which uses a combination of technologies namely, PHP, SPARQL, RDF and jQuery. The application is a cross species phenotype network which allows the fast analysis of the similarity between different phenotypes in organisms, (yeast, fish, worm, fly, rat, slime mold and mouse model) as well as human diseases (OMIM and OrphaNet) The application can be used to find diseases which are related using their phenotypic similarity value.

Contents

1. Background & Objectives

1.1. Background of Project

In the current day and age, the field of Biology has vast amounts of data that they are unable to quantify quickly and as a result they need a system that will enable them to do so. A bridge between the field of Biology and Information Technology is required to provide the tools for this to happen. It requires an understanding and appreciation of both industries, to achieve the optimum results. The project aimed to give Biologists the tools to analyse their data efficiently, and it will directly influence and aid Biologists. Using the application biologists are able to draw upon a huge resource of data, which will be constantly expanding, from all across the globe. The prospect of playing a small part in the discovery of a new understanding or knowledge of a disease or illness, which could lead to providing a better quality of life to people, is a fantastic opportunity. This project has the potential to become a very powerful tool, and one that can be used out in the world for a great purpose.

The major aim of this project is to create a web application which queries an RDF store to provide a fast analysis of the similarity between different phenotypes in animals. This knowledge can then be used to aid our understanding of the function of genes, by systematically analysing the phenotypic outcome of their mutation. This will ultimately facilitate our ability to prioritize causative genes for rare and orphan diseases by enabling the comparison of the phenotypic similarities between experimental data and human clinical signs and symptoms.

The system is available via a web-based platform, so it can be accessed from anywhere with an Internet connection, and the only specifications in terms of hardware or software, is a modern web browser. The web interface is an imperative part of the project, as it provides the tools for a user of any technical ability the opportunity to search the vast amounts of data, by using disease or gene names. This interface forms an essential part of the project since it will allow users to query and analyse the data without requiring knowledge of the query language.

The main technologies being used by the project are PHP and RDF. PHP will be used in collaboration with SPARQL to facilitate the querying of the RDF data stores. The website presentation will be created by using Javascript, XHTML and CSS.

The target audience mostly relates to the Web interface, as the user will never have any dealings with the back-end aspect of the application. In terms of technical ability among users, this could be very vast, and so the project will need to ensure that it can be used by people with even the most basic of computer skills, whilst the same time, it needs to be made in such a way that a skilled user does not find it frustrating. As an example, Google.co.uk, is one of the most easy to use websites on the Internet due to its simplistic homepage. It is obvious for a user that all they need to do is type in a keyword, and they will be provided with a list of results, and this is the type of user experience the project aims to emulate.

In terms of area of expertise, the application will mostly be used by people who have an understanding of Biology or a related field. The application will rarely be used by people who do not have a biological background, as it would not provide any meaningful purpose, and so the terminology that can be used throughout the website can be aimed at this audience.

The main technologies being used by the project are PHP and RDF. PHP is used in collaboration with SPARQL to facilitate the querying of the RDF data stores. The website has been created by using Javascript, XHTML and CSS.

1.2. Related Work

There are a few projects that are similar to this one, from a biological point of view, but are using different technologies. For example, PhenomeNet[1], is a “Cross Species phenotype Network”, which uses a MySQL database to produce a very similar system. PhenomeNet, is the most closely linked to the project as it aims to produce a very similar design for the web application, in terms of simplicity and functionality.

The disadvantage of PhenomeNet is that MySQL is just a database with its own data structure whereas RDF is a W3 standard. An RDF store can be queried directly and remotely using SPARQL, thereby exposing the underlying data to other researchers without them having to parse the dataset themselves. The data will also be available selectively, i.e., only the parts they need.

One of the most pleasing aspects about PhenomeNet is that it is easy for the user to realise where to begin. The homepage provides a quick explanation of the project itself, what it can be used for, and provides a single search box. There is no unnecessary clutter, around the page, and this is feature I would like to try and emulate for the interface of my project. PhenomeNet also has a help page, which explains which key words would be suitable to use, and this will help the user to optimise their search phrases to gain better sets of results. A key feature of PhenomeNet is the 'similarity value', which indicates how similar two diseases are to each other in terms of their symptoms. Other than PhenomeNet there is a lack of competition and rivalry, which means that this tool will serve a great purpose to the biological community.

1.3. Objectives

The aim of the project is to produce a web application, using Web Technologies such as RDF and SPARQL, with an interface that allows a user with very limited computer knowledge to query the RDF store, as well as providing a tool for the analysis of genotypic and phenotypic data. The data learned from the phenotypic data is required for larger computational analysis, and to facilitate these analyses, information must be integrated and accessible in a common format.

In particular it needed to include the following features:

- Ability to search by disease, and gene name.
- View the phenotypes directly and indirectly associated with the selected genotype.

- View the related disease associated with the selected genotype. These include, OMIM, Orphanet, Mouse, Yeast, Worm, Fly, Zebrafish, Slime mold.

- Create a tree structure view of the Mammalian phenotypes for the user to view.

The project utilises an RDF Store, using a suitable data management system. This requires an interaction between the web interface and the data management System, and to do this PHP is used in combination with SPARQL to query the data store. The search terms provided by the user trigger queries to be run on the server and collect the results. The interface displays this data in a meaningful manner to the user, so that they can understand it. Again this ties in with the ease of use and the lack of any technical knowledge on the user's behalf.

2. Development Process

2.1. Introduction

The Waterfall model was originally picked as the chosen methodology due to its rigid nature. One of the key requirements for the Waterfall model to be successful is sufficient planning in the early stages of production, as well as requirement specifications that are not subject to change, and during the initial stages of the project it was obvious that this was lacking. However, despite the requirements being understood, the expected outcome of the project was not too clear which meant that the Waterfall method was not suitable and an Agile method would be more suitable.

This led to changing methodology to Feature Drive Development (FDD) which allows for a more flexible approach, whilst creating working content. The five step process in FDD, fit well with the project as it allows for the initial design plan to be changed as the project progresses. This was a key factor because as the project progressed, the understanding of the tools and functionality of different aspects increased which meant there was the ability to improve the original design.

In the initial stages of the process model, there was an obvious need to amend parts to better suit the project, as FDD and most methodologies are used within a team whereas in this project it would be applied to an individual.

2.2. Methodology and modifications

2.2.1. Develop the application model

The first stage was an overall model of the application, usually referred to as the "Object Model", but as this project does not use object orientated methods, this wouldn't suit it well. Another part of the methodology which had to be amended was the formation of a modelling team, as the project is done by one individual rather than a group of people. The agile nature of FDD meant that despite some aspects being overlooked or not planned for in the original model, such as the Tree Structure, it could be added to the project with ease. Moreover, having only one individual meant that there were going to be a few parts which were overlooked, and so the process accommodated this well. It also meant that as more understanding was gained about the possibilities and limitations of the project, the model could be refined to suit. The overall model is shown in Appendix B.

2.2.2. Build Features List

Based on the application model, the features which would be needed for the application were derived. This was done by splitting the domain up into different areas, in this project each page was used for an area, and under each page a list of features was created. As is consistent with FDD the features should be no more than two weeks in length and if they are they are split up again into smaller parts. Estimating whether a feature could be done within two weeks was difficult as many of the technologies were unknown, and so there was often the need to split features up into smaller parts whilst the development was taking place.

2.2.3. Plan by feature

The majority of this stage did not fit well with this project as it requires the creation of teams and assigning classes to different personnel, however as there was only one individual, none of this was required. What was required was the development plan, which indicates the order the features are created. This became very flexible as there was only one individual, which meant there was no waiting around for other parts of the design, and it was fully understood which parts had already been completed. There was no need to balance the load of different features or put together the planning team as this was impossible.

2.2.4. Design by feature

Once again, there are a number of aspects of this stage which do not suit the individual nature of the project. Firstly the formation of a feature team is not possible, and so there needs to be no discussion as to which parts are needed to be used to create a feature. Also because there is no team, there is no need for the comparison and understanding of different documents which have been produced as everything has been done by one person and so they know everything about the project. During this stage, it becomes clearer what is expected from each feature, and as a result this has a knock on effect to the overall model which needed to be refined each time. This is expected within FDD, and so it is within this stage that the model is expected to be altered, as it would be very rare for a project to not have any changing requirements, or ways of improving the initial design.

The next part of this stage was to create sequence diagrams of the various features of the project. Due to time constraints the diagrams were made a more inclusive than usual, for instance, the pages which contained similar queries could be put under the same sequence diagram. These are displayed in Appendix C, D and E of this document. Sequence Diagram 1, shows the process of searching for a disease using the application, and Sequence Diagram 2 shows the process of using the tree structure to search for a disease related to the phenotype selected. Sequence Diagram 3, shows how a query is ran and which services it has to connect to in order to give the user the feedback they require.

2.2.5. Build by feature

The final stage of the methodology was implementing all of the required aspects to achieve the requirements set by the feature list. Because the project didn't have multiple members, features were not always constructed at the same time, and rather they were built in a more chronological approach. However, it was still common for spike work to be on going on different features, such as the creation of the parse scripts which often took a few days to run.

The benefit to this approach was that every time a feature was developed, it was possible to operate on a standalone basis, or as part of the system. This meant each feature didn't have a large impact upon other aspects of the system, and so it was easy to build them without having to have much knowledge of the other features. Obviously there needed to be some interactions between the different features, although the most complex this got was connecting to the SPARQL endpoint from the PHP page, which was only one line of code.

Creating a feature is usually a quick task, as mentioned above it should take no more than two weeks per feature. During this period a means of tracking the current progress made on a feature was required. In FDD this is done by the use of milestones, and a table looks like this:

Domain Walkthrough	Design	Design Inspection	Code	Code Inspection	Promote to build.
1%	40%	3%	45%	10%	1%

This table refers to the last two stages of the process, "Design by Features" and "Build by Features", and the figures used above are consistent with any feature. If a feature has only got half way through the Design stage its overall progress can only be measured at 1%, not 21%. This can be seen as quite a pessimistic way to view progress, but as it is consistent with every single feature on every project, it provides valuable feedback to the project leader who gets an accurate representation of which stages are yet to be completed. Using this, it is also possible to check if a feature was completed in less than two weeks, or if the feature needs to split up further. Despite this being quite late in the overall process, the agile nature of FDD makes this possible, and often encouraged.

2.2.6. FDD coloured UML.

This class diagram can be seen in Appendix K, and is explained in the Design part of this document.

2.3. Impact of the Process Modification

The biggest impact on the Process Model was applying it to a project which only had one member. As mentioned numerous times above, this meant a lot of stages had to be adapted and changed, as well as assigning all duties to a single person. The individual will always know everything about the project, e.g. what is happening and what needs to happen next, so the overall progress is always understood. As a result, the milestone tables become expendable as it would waste time to constantly be updating and creating tables which would never be read by anybody other than the individual.

The individual nature of the project also had an impact on the testing which is usually performed during FDD. Code Reviews, are usually performed by a team who provide feedback on other member's work and offer feedback and suggestions. As a result none of this could be achieved and it had to be reworked to suit an individual testing on their own. This is discussed in more detail in the Testing section of this document.

The Process Model was also adopted for use on a Web application, rather than an Object Orientated piece of software. This meant that there it was difficult to perform unit tests, and also meant that the

Overall Model showed the flow of data and structure of the application rather than the interactions between different classes as this was not applicable in this project.

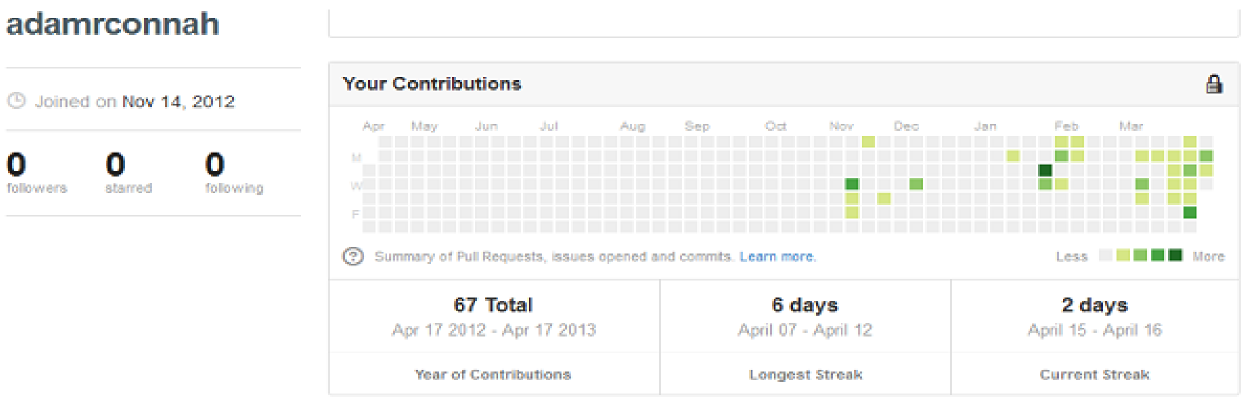
2.4. Choice of implementation tools.

2.4.1. GitHub

One tool used heavily throughout the project was GitHub, which is a piece of software used for version control during projects. It operates in a similar fashion to many version control systems, using similar terminology such as 'commit'.

The biggest feature GitHub provides is the backing up of files to the cloud service provided by the company. This means that if the local data is corrupted, a new branch can be quickly downloaded and with minimum disruption to any work. GitHub also allows for older versions of documents to be viewed and accessed, which enables a side by side comparison between old and new files. This can be useful when a piece of code that was previously working starts to produce errors, as the changes can be viewed which allows for quicker debugging. GitHub was used throughout the project to the end, providing no extra delay or hindrance to the project as it was very easy to learn to use. One of the most pleasing factors was the cross platform capabilities, as the project was often being worked upon from either a Windows or Linux base, and so the documents needed to be accessible from both systems.

GitHub also provides different ways of using the service, either via the desktop application, the Shell Command, or the browser application. The desktop application is very useful for first time users of the service, as it very simple and allows for the basic functions to be performed quick and easily. This is what was used at first, whilst getting used to GitHub and its capabilities, as well as getting used to Version Control in general. The Shell Command line was trickier to use, and at first the documentation on the GitHub website came in very useful. After extended use the Shell Command became very easy and fast to use, and it offered more capabilities than the Desktop Application. The Browser Application is different to the two other means of working with GitHub, and isn't used as a place where files are constantly committed from. Instead it is used to view the data currently held, and it provides huge amounts of details about this data, such as number of commits, and a heat graph showing the dates on which commits have occurred. Below is the heat map of this project, correct as of 17/04/13.


adamrconnah

Joined on Nov 14, 2012

0 followers 0 starred 0 following

Your Contributions

Summary of Pull Requests, issues opened and commits. Learn more.

67 Total	6 days	2 days
Apr 17 2012 - Apr 17 2013	April 07 - April 12	April 15 - April 16
Year of Contributions	Longest Streak	Current Streak

The GitHub link for this project is: https://github.com/adamrconnah/MajorProject

2.4.2. Dropbox

Another tool used during the project was Dropbox, which is a file hosting service available across multiple platforms. In contrast to GitHub, Dropbox is not a version control piece of software, and does not create backups of older versions of files. However, during the project it was not needed for that as GitHub was already employed for this task. Instead Dropbox was primarily used as a secondary back up to the GitHub directory on whichever machine was being used. This meant that the files could be processed outside of the GitHub directory, and could consequently be moved to the GitHub directory, and from there they would be committed. This was especially handy whilst working on the University Computers, as any GitHub directory would be placed on the D: drive, and when the computers are restarted they remove all data from this drive. This meant that even if the file was being saved as it was being worked on, it would all be lost if the computer encountered an error. A Dropbox repository was no different, in terms of it being lost if the computer shut down, but in contrast to GitHub, Dropbox constantly syncs with the server after each save which means that it was less likely to lose data in the event of an error.

2.4.3. RDF

RDF was recommend for the project by the project supervisor, but there still needed to be a consideration of why the use of RDF is better than traditional methods such as XML. The main advantage that this system has over its competitors is the use of 'RDF'(Josh Tauberer, 2005)(Resource Description Framework). Using an RDF store is key to the project as it is a decentralized system, and hence it facilitates the retrieval and analysis of data that resides in a variety and diverse sources. Such a store generates a large resource that can be accessed quickly and efficiently providing biomedical researchers and bio-informaticians a valuable resource for their analysis.

RDF is, in many ways, an extension of traditional XML, and many similarities can be drawn from its syntax as well as its application. RDF is designed to be read and understood by humans, and not for a machine, unlike XML. However, the difference between the two technologies is that XML is very much a 'data format' with 'no built in meaning' (Dan Zambonini, 2005). RDF is different because it represents the data like a model and with that it means that systems can gain a lot more knowledge from what they are seeing.

With XML we can write a programme which links many tables of data together, but this would be very tiresome and strenuous. The standards of RDF provide the ability to create these systems in a decentralized environment. Another benefit of using RDF is that it can accommodate the mixing of vocabularies.

"A vocabulary about TV shows developed by TV aficionados and a vocabulary about movies independently developed by movie connoisseurs must be able to be used together in the same file, to talk about the same things, for instance to assert that an actor has appeared in both TV shows and movies." (Joshua Tauberer, 2005).

With RDF it is very easy to link together two or more files, and arriving at conclusions that each file on its own couldn't represent. This is possible because all RDF files can use any chosen Vocabulary, which allows for huge flexibility and even bigger resources of information to be accessed. An RDF store can also be used to expose the data as 'Linked Data' (Tom Heath, No Date) and reference other data sources (drugs, proteins, animal models, diseases). Using SPARQL queries, it even becomes possible to run complex queries which span multiple endpoints, often in various domain locations from various providers.

Tim Berners Lee points out a flaw with XML, when he notes the following. "Without looking at the schema, you know things about the document structure, but nothing else. You can't tell what to deduce." (Tim Berners Lee, 1998) XML heavily relies on the reader knowing the schema to be able to have a full understanding of the data, and can become very messy when it comes to querying in comparison to the very simple nature of RDF.

RDF by its very nature was designed to be shared amongst applications, intended for people or applications that didn't create it. All data stored in RDF has some meaning and this is presented by means of "Triples", which are the foundation of RDF, so much so that an RDF store is often referred to as a Triple store. Triples, sometimes referred to as statements, always consist of a Subject, Predicate and Object. For example Alzheimer (subject), MP:000001 (object), "has a" (predicate) tells us that Alzheimer has a phenotype with an ID MP:000001. In RDF this is known as a "fact", and by putting a lot of these together we can start to build a knowledge base. After the consideration of all the factors named above, it is clear to see the benefits which RDF offers in an environment where data is created to be shared.

2.4.4. PHP

PHP is primarily a scripting language and so it serves as the perfect candidate to be used when creating the parse scripts as well as other aspects of the website. PHP is open source, and as a result has a huge amount of community support, as well as having thousands of features, with more and more being added regularly. PHP offers a big advantage with the ability to embed it inside HTML code, as well as producing HTML code itself. Before beginning the project it was unknown which languages would work well with SPARQL, but after research there were a number of libraries which assisted the interaction between the two technologies, these are explained later on in the document, and this led to the adoption of PHP for the project.

2.4.5. jQuery

jQuery (Sheo Narayan, 2011) is a Javascript library designed to allow a more simple approach to client side scripting. Like PHP it is also open source, and has huge amounts of documentation on its website, which meant that despite having no knowledge of jQuery before using it, it became very clear which function to use in the appropriate situation. The main advantage of jQuery is its ease of use, as it requires a lot less amounts of code to achieve the same results as Javascript would need. Another feature of jQuery which made it suitable for the project was its Ajax support, which facilitated the creation of the Tree structure.

2.4.6. Virtuoso

The choice of Virtuoso as the data management store is explained in detail in the implementation section of this report, as it also outlines some problems with the software.

3. Design

This section lists all files used in the Web Application and discusses their purpose, interactions, and what is happening within the code as well as providing a general overview of the architecture of the application.

3.1. Overall Architecture

The Web application is quite simple in how it operates:

- A keyword is entered into the HTML form on the web site.
- This variable is then sent via the POST method to the page "diseases.php".
- Here, the PHP code uses the RAP RDF API library to connect to the SPARQL endpoint Virtuoso.
- The variable is used in the SPARQL query in the diseases.php script.
- The results of this query are displayed on the page.
- The user is then given two choices. View related genotypes and diseases, or related phenotypes.
- If they choose the former, then the variable will be taken from whichever entry was selected, and sent using the GET method, to the edge.php page.
- Here the script operates similarly to the diseases.php script, with a few different filter options in place, and provides a similarity value.
- If they user had chosen the latter option, the variable would be taken from the entry selected and send using the GET method to the ontologyterms.php page.
- Here the script does a similar job as the other scripts, and returns a list of related phenotypes.

The Tree Structure performs in a different manner, as it uses AJAX to produce the required outputs.

- When the page loads, the jQuery script gets the first set of children from the treescript.php
- Each time a link is clicked, the jQuery performs the same operation, except this time it sends a variable via GET, and receives the children of that variable.
- If the link is clicked again, the children are contracted.

3.2. An Explanation of the Overall Model

The model, shown in Appendix B, displays a hierarchical view of the application as well as the way information flows between files, which gives an overall understanding of what the application requires. This model draws inspiration from the flow diagram which was constructed in the very early stages of the project, which can be seen in Appendix H. On the overall model the item marked with "1." on the model represent the homepage of the web application which will be the first page users visit when they load the website, and all items marked with "2." are very simple pages which either provide help or a contact form.

All items marked with “3.” are responsible for creating the tree structure in the application, and work very closely, with information being constantly passed to each other, to provide the final result. The item marked with a “7.” represents the RDF store, which is what will hold all of the data for the application, and will need to be accessible by all of the pages which perform queries. Finally, items marked with, “4, 5, 6” perform SPARQL queries which will be the main focus of the application. Each individual component is explained more in the detailed design section.

3.3. Detailed Design

3.3.1. Feature List

The feature list was created at the start of the project, but has been amended at different points.

Below is the feature list for the project:

- Scripts which parse the data, and create RDF data.
- A Data management system to store the RDF data.
- A web page which searches for all diseases.
- A web page which displays all related diseases.
- A web page which displays all related phenotypes.
- A web page with tree structure to explore phenotypes.
- Contact form which sends an email to selected address.

3.3.2. Coloured UML diagram.

In Appendix K is the coloured UML diagram, the diagram helps to explain the structure of the application, with more emphasis on the variables. Because the application is not object orientated it cannot be solely relied upon for the whole design of the application, however it is still useful for the project as it gives a model representation of the interactions throughout the system.

The model in this case quickly shows that a lot of the pages in the application are very similar in design, as well as the way they interact with each other. It is easy to see which pages will require a different approach and which can re-use code from other areas.

The key for the diagram is as follows:

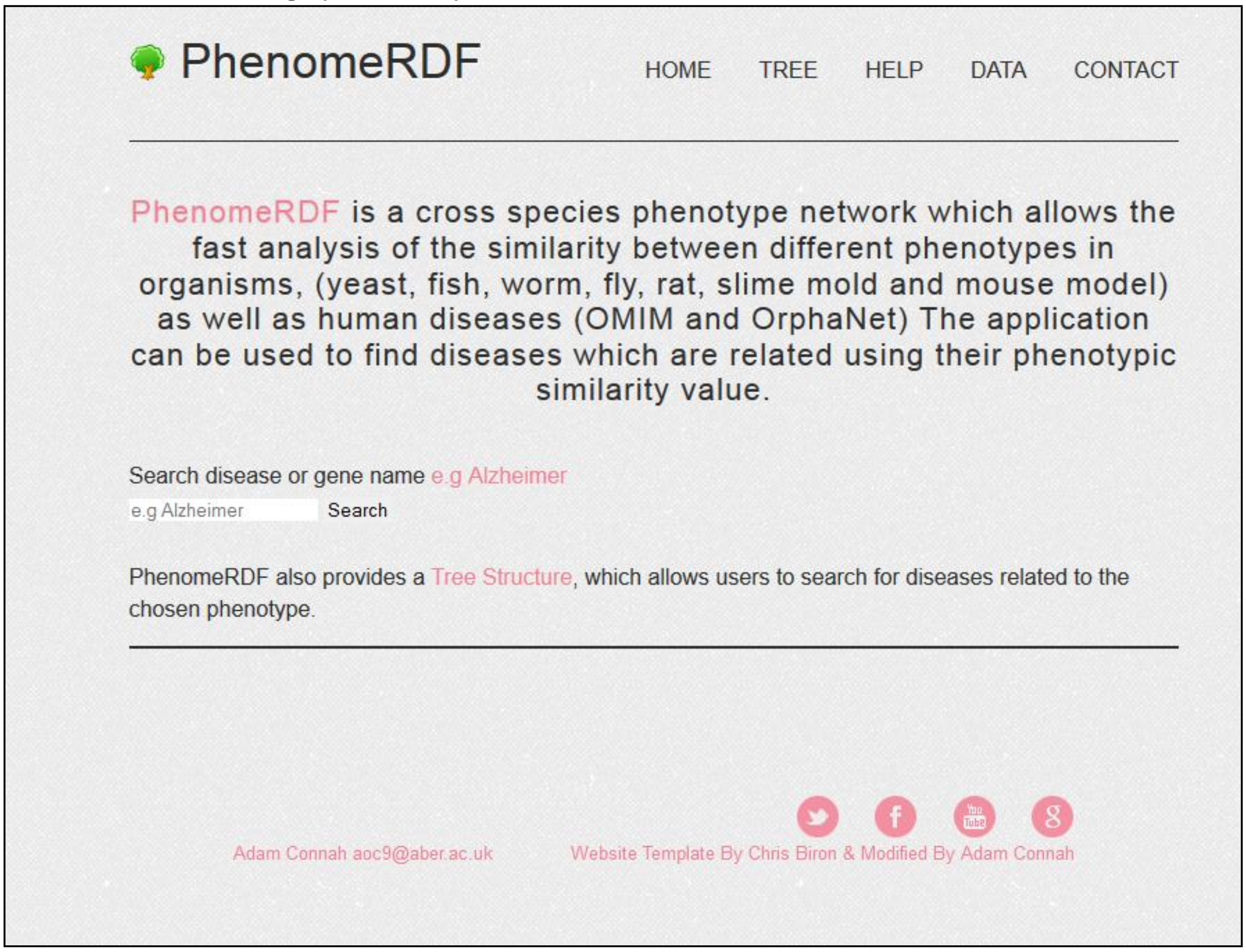
Yellow: a role in the system.

Blue: a reusable description.

Green: an entity.

Pink: a moment or interval of time. (Neil Taylor, 2012)

3.3.3. Home Page (index.html)



PhenomeRDF is a cross species phenotype network which allows the fast analysis of the similarity between different phenotypes in organisms, (yeast, fish, worm, fly, rat, slime mold and mouse model) as well as human diseases (OMIM and OrphaNet) The application can be used to find diseases which are related using their phenotypic similarity value.

Search disease or gene name e.g Alzheimer

e.g Alzheimer Search

PhenomeRDF also provides a Tree Structure, which allows users to search for diseases related to the chosen phenotype.

Adam Connah aoc9@aber.ac.uk Website Template By Chris Biron & Modified By Adam Connah

Figure 1. Index.html

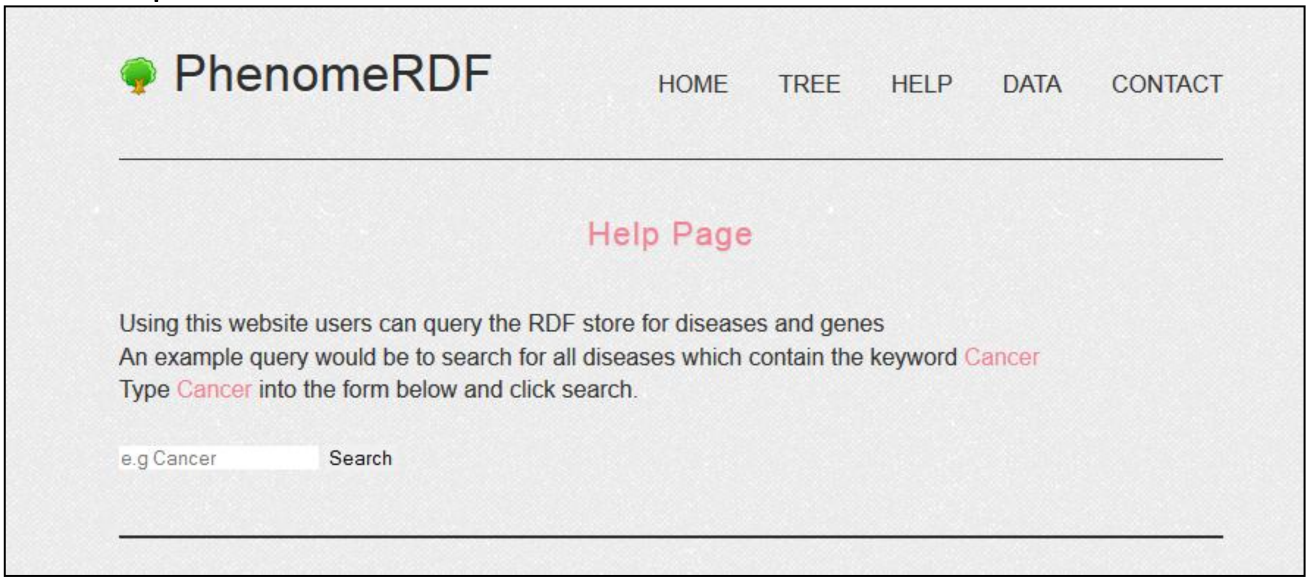
Index.html is the Home Page for the website, and its initial purpose is to describe the function of the site. It must be easy for the user to find the information they require quickly otherwise the user may get frustrated and leave the website. The homepage aims to answer the following questions:

- What does our company/organisation do?
- What can users do at our website?
- What is the purpose of the site?

With this page, the user can quickly read the synopsis of the application and understand what it service it provides. If this meets the user's need then it is fairly evident where to start with the Search Query form in the centre of the page. Users can also navigate to other pages of the website using the menu in the top right hand corner.

This page is fairly simple in design and it only uses CSS and HTML to create it the forms and layout, as well as a small amount of Javascript to ensure that the form is not empty. Index.html interacts with all but one page using hyperlinks; the exception is the relationship with diseases.php. This interaction uses the PHP method of POST to send the data entered into the form to diseases.php where it is processed.

3.3.4. Help.html



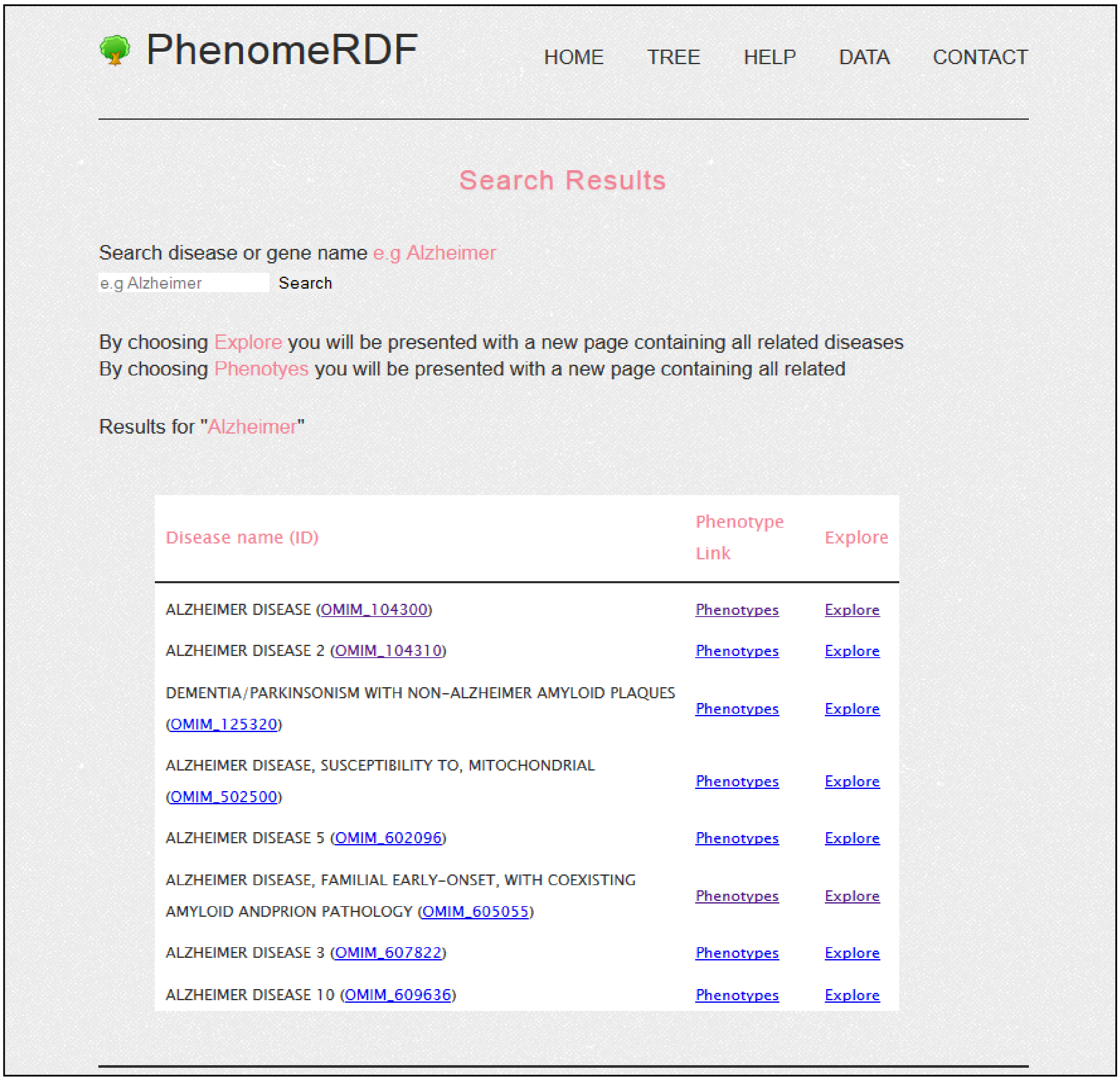
Help Page

Using this website users can query the RDF store for diseases and genes
An example query would be to search for all diseases which contain the keyword Cancer
Type Cancer into the form below and click search.

Figure 2. Help.html

This is a very simple page which provides the user with some assistance to users who may not understand what the purpose of the application is.

3.3.5. Diseases.php



Search Results

Search disease or gene name e.g Alzheimer

e.g Alzheimer Search

By choosing Explore you will be presented with a new page containing all related diseases
By choosing Phenotyes you will be presented with a new page containing all related

Results for "Alzheimer"

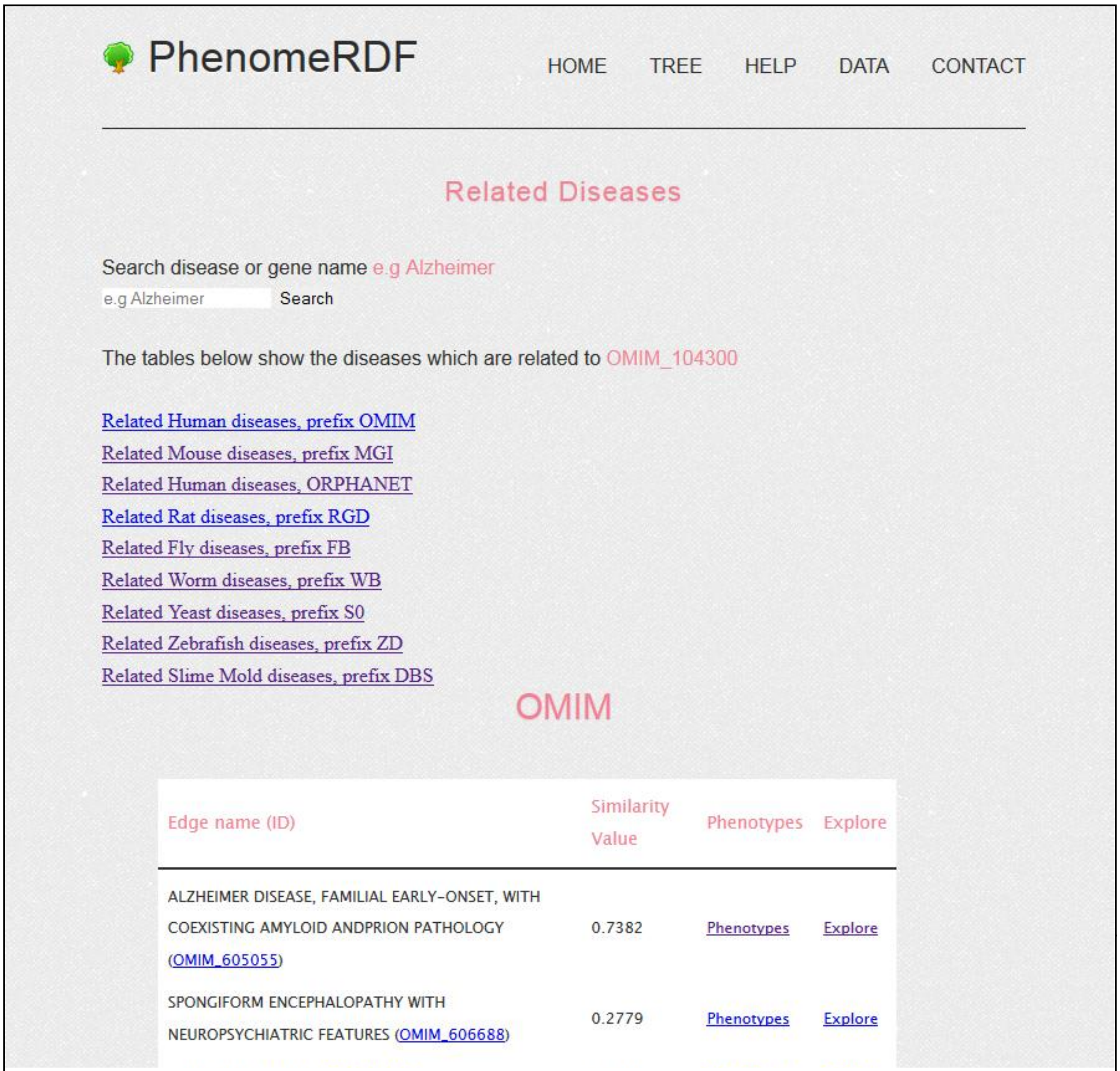
Disease name (ID)	Phenotype Link	Explore
ALZHEIMER DISEASE (OMIM_104300)	Phenotypes	Explore
ALZHEIMER DISEASE 2 (OMIM_104310)	Phenotypes	Explore
DEMENTIA/PARKINSONISM WITH NON-ALZHEIMER AMYLOID PLAQUES (OMIM_125320)	Phenotypes	Explore
ALZHEIMER DISEASE, SUSCEPTIBILITY TO, MITOCHONDRIAL (OMIM_502500)	Phenotypes	Explore
ALZHEIMER DISEASE 5 (OMIM_602096)	Phenotypes	Explore
ALZHEIMER DISEASE, FAMILIAL EARLY-ONSET, WITH COEXISTING AMYLOID ANDPRION PATHOLOGY (OMIM_605055)	Phenotypes	Explore
ALZHEIMER DISEASE 3 (OMIM_607822)	Phenotypes	Explore
ALZHEIMER DISEASE 10 (OMIM_609636)	Phenotypes	Explore

Figure 3. diseases.php

Diseases.php is a bit more complex in design and functionality than the homepage, and is the first page a user will be presented after searching a keyword. The job of this page is to provide the first set of search results to the user. These results are presented in three columns, disease Name (ID), Explore and phenotypes Link. Under disease Name (ID), the results from all of the disease names which matched the keyword provided by the user. Coupled with this is the ID presented as a link so that the user can visit the appropriate Genome website page specific to the disease selected. The Explore column provides the user with a link to the edge.php page, where they can view diseases related to the one they have selected. Similarly the phenotypes column enables the user to see all Human and Mammalian phenotypes related to the disease selected, by interacting with ontologyterms.php. Both of these interactions are done using the GET method in PHP, as it doesn't need to be secure and it can also allow more experienced users to edit the URL to query for different results without having to look for the link they require.

To produce the desired output the page uses a mixture of technologies, HTML, CSS, PHP and SPARQL. When it is first loaded, the page is built using HTML and CSS, it then interacts with index.html and receives a variable using PHP via the POST method. PHP is again used, this time in conjunction with the RAP RDF API library to perform a SPARQL query. The query connects to the endpoint biocrunch.dcs.aber.ac.uk:8890/sparql, where the RDF store has been created and searches the relevant graph. The query then specifically searches for all diseases which have a name, where the name is equal to the variable received from the POST method. All of the results are assigned to PHP variables, and are printed and tidied up using various aspects of PHP functions to produce the table of results.

3.3.6. Edge.php



Related Diseases

Search disease or gene name e.g Alzheimer

e.g Alzheimer

The tables below show the diseases which are related to OMIM_104300

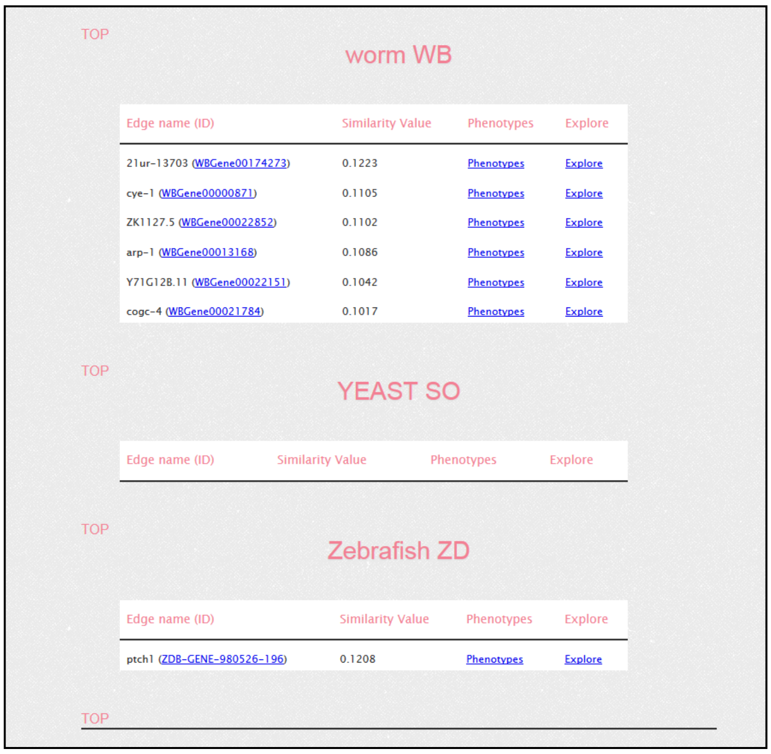
Related Human diseases, prefix OMIM
Related Mouse diseases, prefix MGI
Related Human diseases, ORPHANET
Related Rat diseases, prefix RGD
Related Fly diseases, prefix FB
Related Worm diseases, prefix WB
Related Yeast diseases, prefix S0
Related Zebrafish diseases, prefix ZD
Related Slime Mold diseases, prefix DBS

OMIM

Edge name (ID)	Similarity Value	Phenotypes	Explore
ALZHEIMER DISEASE, FAMILIAL EARLY-ONSET, WITH COEXISTING AMYLOID ANDPRION PATHOLOGY (OMIM_605055)	0.7382	Phenotypes	Explore
SPONGIFORM ENCEPHALOPATHY WITH NEUROPSYCHIATRIC FEATURES (OMIM_606688)	0.2779	Phenotypes	Explore

Figure 4. edge.php

Edge.php is very similar to diseases.php in terms of interactions and technologies. It also built using HTML, CSS, PHP and SPARQL, and when it is loaded receives a variable from using the GET method in PHP. This variable however is not limited to where it comes from, and can be sent from either diseases.php or itself. Its purpose however is quite different as it uses the RDF data to find all diseases that are similar to the variable provided.


TOP

worm WB

Edge name (ID)	Similarity Value	Phenotypes	Explore
21ur-13703 (WBGene00174273)	0.1223	Phenotypes	Explore
cye-1 (WBGene00000871)	0.1105	Phenotypes	Explore
ZK1127.5 (WBGene00022852)	0.1102	Phenotypes	Explore
arp-1 (WBGene00013168)	0.1086	Phenotypes	Explore
Y71G12B.11 (WBGene00022151)	0.1042	Phenotypes	Explore
cogc-4 (WBGene00021784)	0.1017	Phenotypes	Explore

TOP

YEAST SO

Edge name (ID)	Similarity Value	Phenotypes	Explore

TOP

Zebrafish ZD

Edge name (ID)	Similarity Value	Phenotypes	Explore
ptch1 (ZDB-GENE-980526-196)	0.1208	Phenotypes	Explore

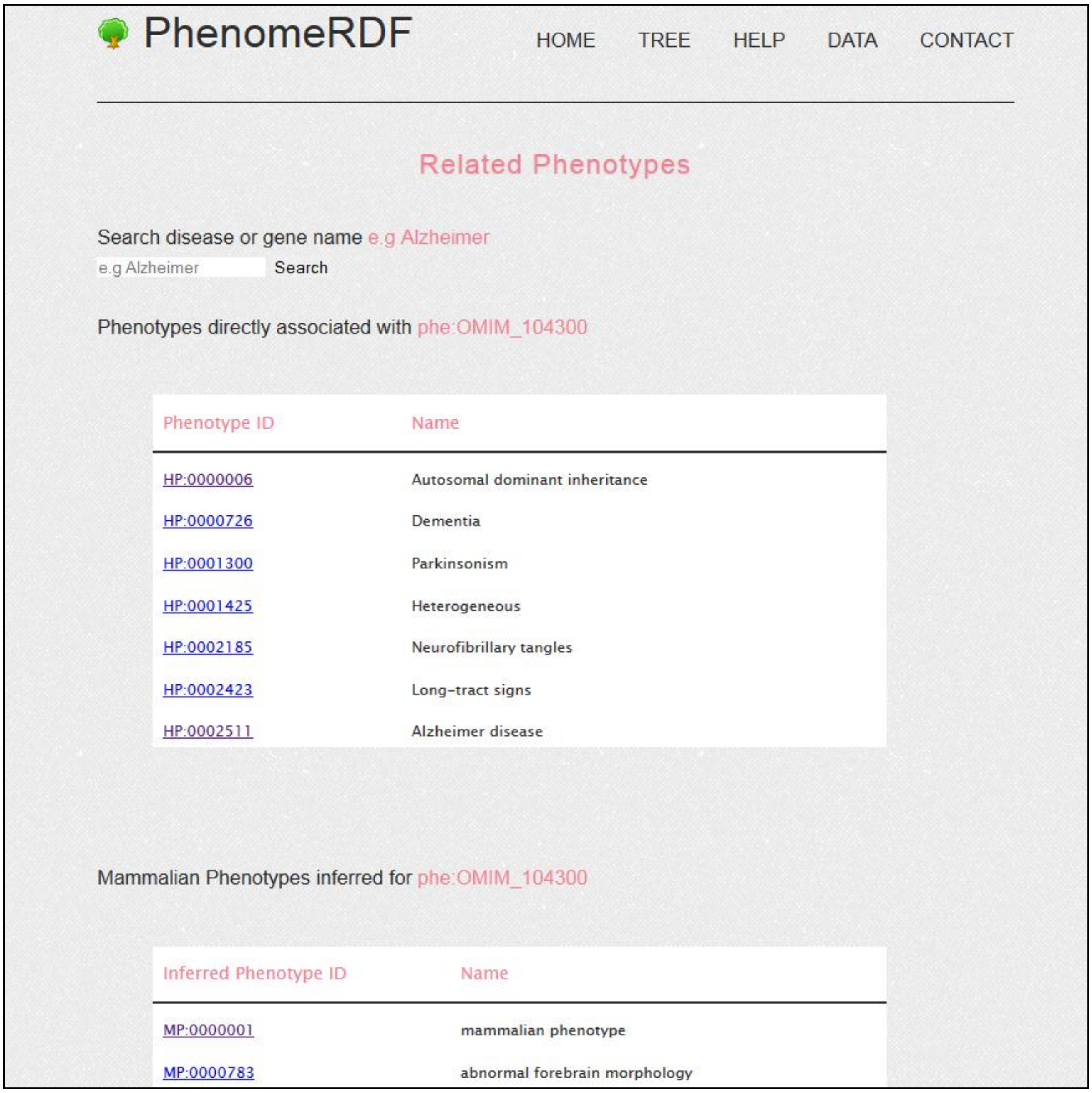
TOP

Figure 5. edge.php (bottom of page)

The table of results has four columns, three of which have the same functionality as they did on the diseases.php page. These are Edge Name(ID) which is identical to the disease Name (ID) column, and phenotype Link and Explore. The added column Similarity Value, is the key to making the comparison between all of the stored data possible, as all diseases have a relationship, however small, and this allows the application to display only the diseases that have a significant relationship.

Overall the whole process of this page is very similar to diseases.php as it uses PHP to connect to the data store and query using SPARQL, the main difference is that there are nine queries performed on this page. All of the queries on this page are very similar: Firstly they search for all diseases that have a related disease, a value, and a name. The query then throws away any results that are matched to itself, and returns only the results that have a similarity value of greater than 0.1. The final step is to filter the results into different categories (shown in Figure 5), so that the user can quickly navigate to each different table depending on what they require, shown in the. This is the only reason there is multiple queries, if the data set was smaller, it might be okay to put all results in the same table, but it is often the case that there are a lot of results, and this make it is easier for the user.

3.3.5. Ontologyterms.php



Related Phenotypes

Search disease or gene name e.g Alzheimer

e.g Alzheimer

Phenotypes directly associated with phe:OMIM_104300

Phenotype ID	Name
HP:0000006	Autosomal dominant inheritance
HP:0000726	Dementia
HP:0001300	Parkinsonism
HP:0001425	Heterogeneous
HP:0002185	Neurofibrillary tangles
HP:0002423	Long-tract signs
HP:0002511	Alzheimer disease

Mammalian Phenotypes inferred for phe:OMIM_104300

Inferred Phenotype ID	Name
MP:0000001	mammalian phenotype
MP:0000783	abnormal forebrain morphology

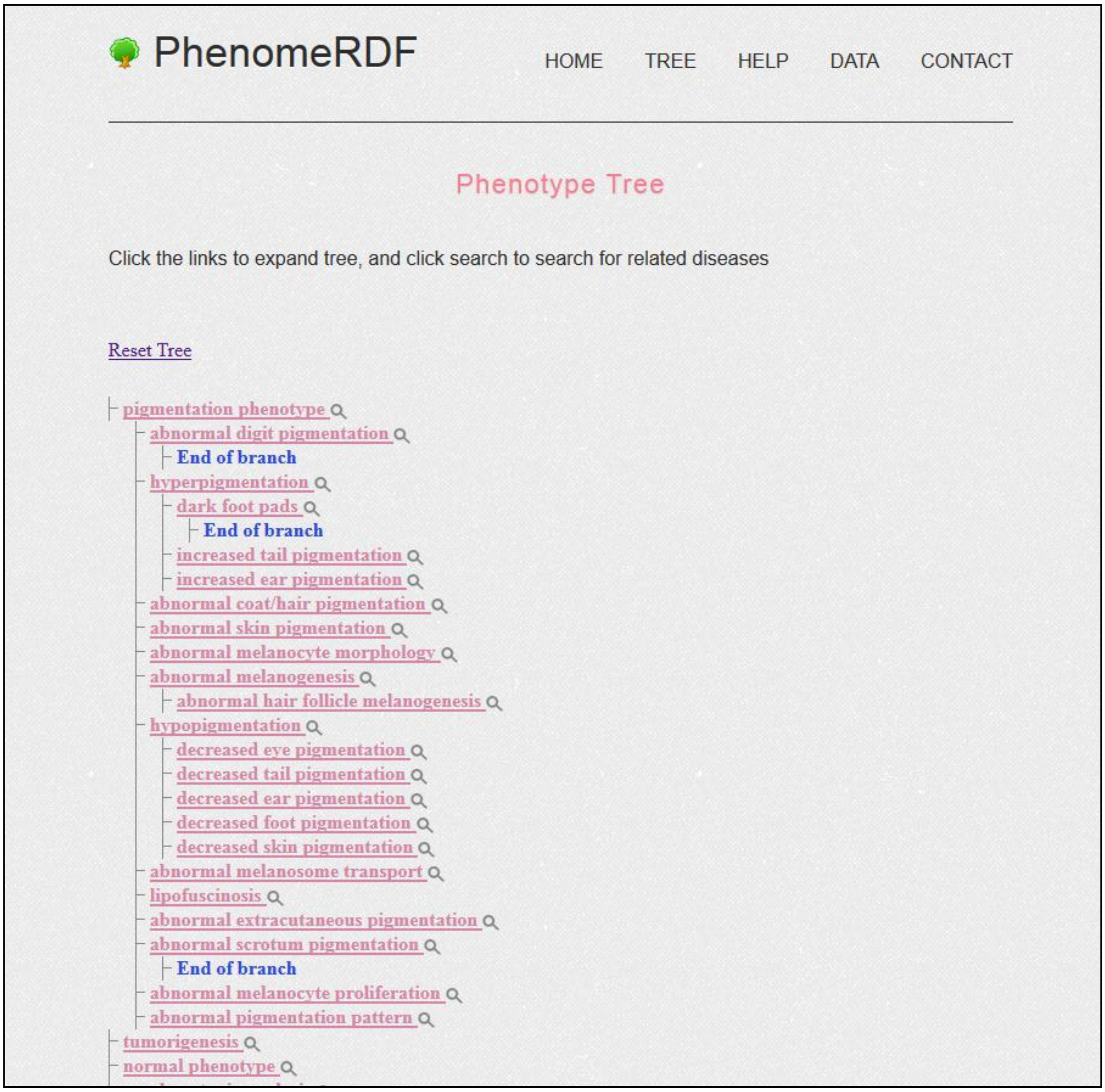
Figure 6. Ontologyterms.php

This page is again very similar to diseases.php and edge.php with regard to the functionality, the difference being the data that has been pulled from the data store. Ontologyterms.php is also built using HTML, CSS, PHP and SPARQL, and receives variable via the GET method in PHP. The purpose of the page differs however, and its job is to provide all of the related Human and Mammalian phenotypes in relation to the variable that it received.

The table of results differs to the other results pages on the web site quite a lot, with only two columns: phenotype ID (Inferred phenotype ID in the table below), and Name. The phenotype ID is a hyperlink to external sites which provide more information about the selected phenotype, whereas the Name column just provides the common name given to the phenotype. The page only interacts with edge.php to receive the variable, and doesn't pass any data to other pages on the web site.

The process is once again very similar to the other pages as it uses the same techniques to connect to the RDF store and performs queries in the same manner. The page has two queries, and uses the variable sent from edge.php to perform them. The first searches for all of the phenotypes associated with this disease, and the second searches for all of the inferred phenotypes associated with the disease.

The main problem with this page is that it leads to a dead end and the user will either end up outside of the website or have to start a new query. However, as the whole purpose of the application is to provide users with a platform to find the data they require, and to do this they must be passed to an external site which holds more data about the specific element they have chosen to explore.



Phenotype Tree

Click the links to expand tree, and click search to search for related diseases

Reset Tree

```
├ pigmentation phenotype
  ├ abnormal digit pigmentation
    ├ End of branch
  ├ hyperpigmentation
    ├ dark foot pads
      ├ End of branch
    ├ increased tail pigmentation
    ├ increased ear pigmentation
  ├ abnormal coat/hair pigmentation
  ├ abnormal skin pigmentation
  ├ abnormal melanocyte morphology
  ├ abnormal melanogenesis
    ├ abnormal hair follicle melanogenesis
  ├ hypopigmentation
    ├ decreased eye pigmentation
    ├ decreased tail pigmentation
    ├ decreased ear pigmentation
    ├ decreased foot pigmentation
    ├ decreased skin pigmentation
  ├ abnormal melanosome transport
  ├ lipofuscinosis
  ├ abnormal extracutaneous pigmentation
  ├ abnormal scrotum pigmentation
    ├ End of branch
  ├ abnormal melanocyte proliferation
  ├ abnormal pigmentation pattern
├ tumorigenesis
├ normal phenotype
```

Figure 7. tree.html

3.3.6. Tree.html

The overall purpose of the page is to provide a tree structure containing all phenotypes and their children, which can be expanded and contracted at the control of the user. The user can then search all diseases related to the phenotype that they select.

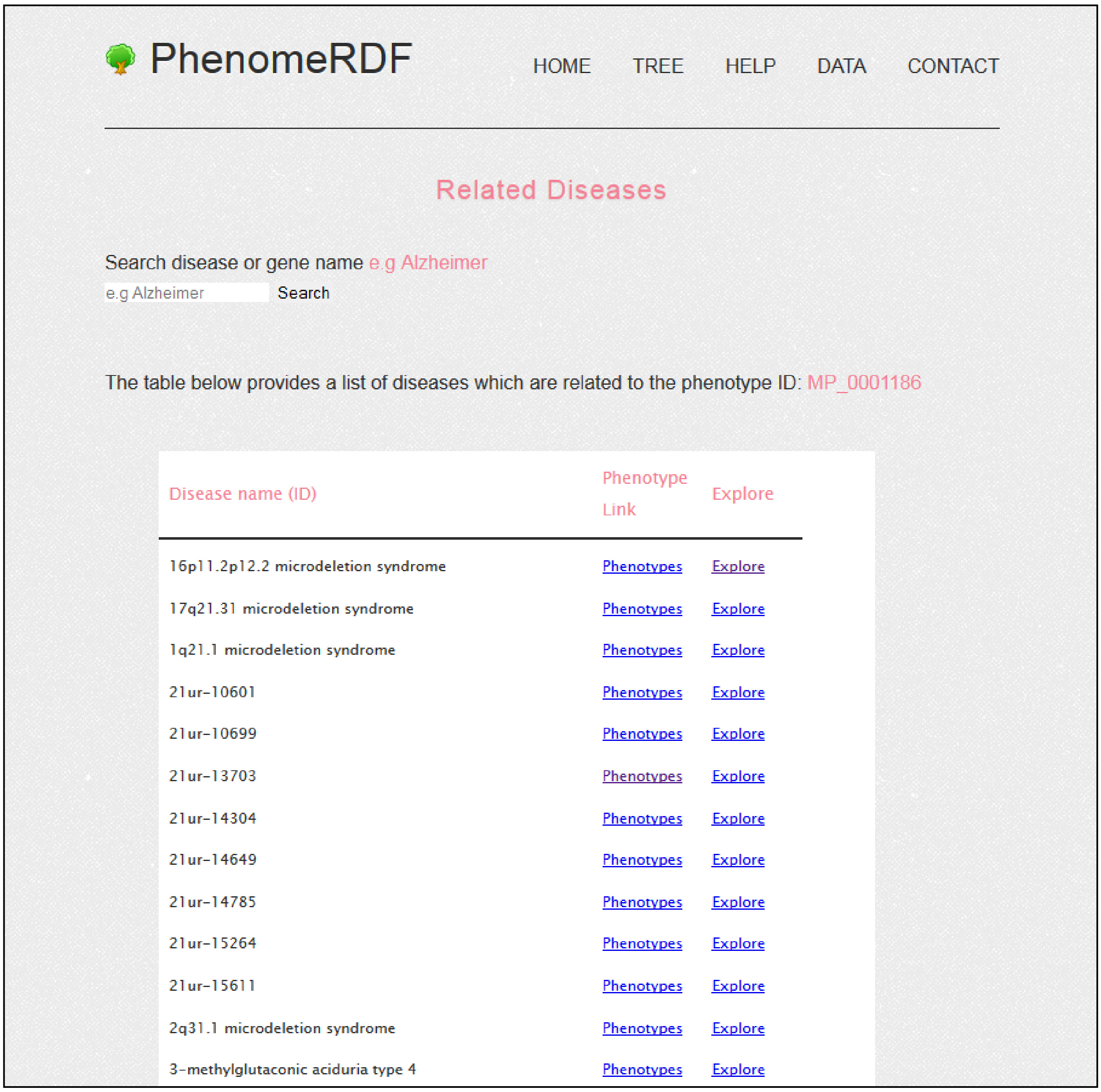
This page is completely contrasting to any other page on the web site in both its layout and function. The page also uses different technologies, with the use of jQuery to produce dynamic content on the page using AJAX techniques to get the necessary data from treescript.php, along CSS and HTML to build the page. This page heavily relies on treescript.php and the interaction between the two is key to the desired outcome.

3.3.7. Treescript.php

Treescript.php is where the processing is done to make the tree structure possible. It runs at the request of tree.html, and uses PHP to produce the HTML that is displayed on tree.html. It also interacts with the mammalian_phenotype.obo file which contains the Ontology of Mammalian phenotypes.

Treescript is dependent on the variable it is fed before it runs, and the default value is the Root of the mammalian_phenotype.obo, which is MP:0000001, and by using this variable it can determine all of the phenotypes which have a relation to it. If the script receives a variable from elsewhere, it can perform the same process and find the children of the given ID. The overall technique is quite simple, as it looks for all entries in the mammalian_phenotype.obo file which say that they have a relation to the given variable ID, and sends them in a formatted manner to the user.

3.3.8. phenotypes.php



Related Diseases

Search disease or gene name e.g Alzheimer

e.g Alzheimer Search

The table below provides a list of diseases which are related to the phenotype ID: MP_0001186

Disease name (ID)	Phenotype Link	Explore
16p11.2p12.2 microdeletion syndrome	Phenotypes	Explore
17q21.31 microdeletion syndrome	Phenotypes	Explore
1q21.1 microdeletion syndrome	Phenotypes	Explore
21ur-10601	Phenotypes	Explore
21ur-10699	Phenotypes	Explore
21ur-13703	Phenotypes	Explore
21ur-14304	Phenotypes	Explore
21ur-14649	Phenotypes	Explore
21ur-14785	Phenotypes	Explore
21ur-15264	Phenotypes	Explore
21ur-15611	Phenotypes	Explore
2q31.1 microdeletion syndrome	Phenotypes	Explore
3-methylglutaconic aciduria type 4	Phenotypes	Explore

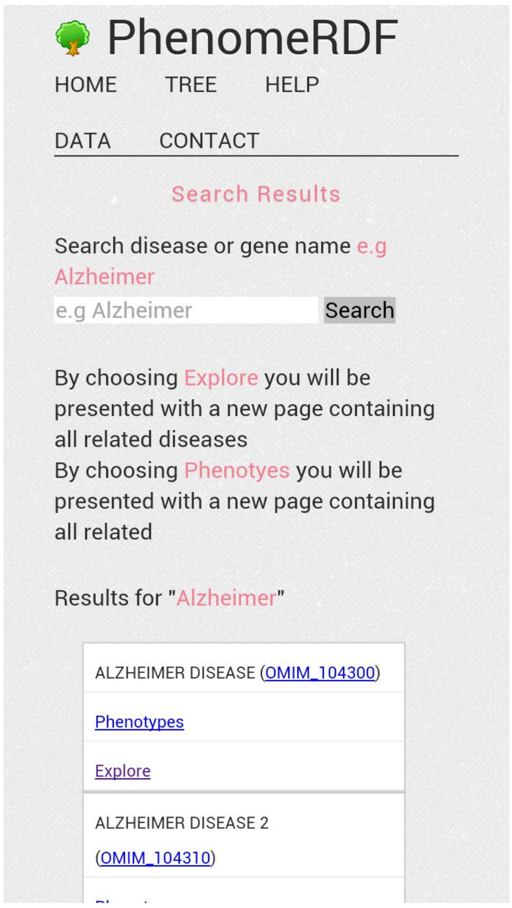
Figure 8. phenotypes.php

This page is the page the user will be sent to after using the search function on the Tree Structure and here they will be presented with a table of results relating to the phenotype they chose. It is very similar to diseases.php and the table of results is set out in identical fashion with the three column headings disease name (ID), Explore and phenotype Link and the perform the same actions as they do on diseases.php. The page also uses the same technologies to build the page HTML, CSS, PHP and SPARQL.

phenotypes.php interacts with edge.php as well as ontologyterms.php by sending the disease name (ID) across as a variable via the GET method in PHP, and also interacts with tree.html when receiving the phenotype ID also via the GET method. The key difference between this page and diseases.php is the variable it receives and this means that the queries need to be done differently.

The SPARQL query on this page searches for all diseases that have an inferred phenotype which matches the ID of the one given from tree.html, and lists all results that satisfy the parameters.

3.4. Web Interface Design


PhenomeRDF

HOME TREE HELP

DATA CONTACT

Search Results

Search disease or gene name e.g Alzheimer

e.g Alzheimer Search

By choosing Explore you will be presented with a new page containing all related diseases
By choosing Phenotyes you will be presented with a new page containing all related

Results for "Alzheimer"

ALZHEIMER DISEASE (OMIM_104300)

Phenotypes

Explore

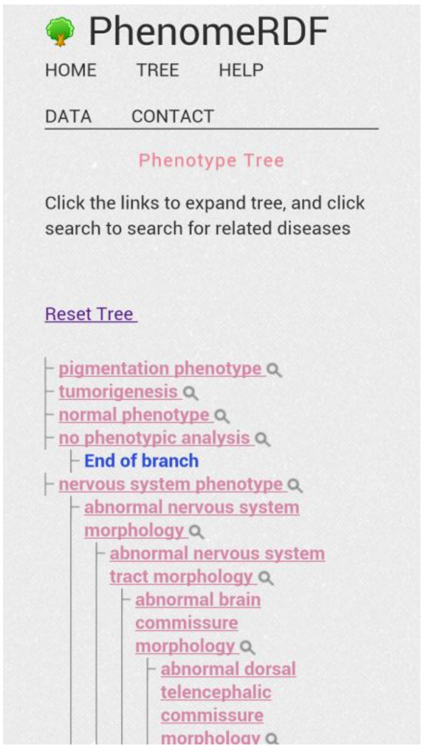
ALZHEIMER DISEASE 2 (OMIM_104310)

Figure 9. Mobile view of diseases.php (Portrait)

The main influences for the Web Interface, were from Google and PhenomeNet (Robert Hoehndorf, Paul N. Schofield and Georgios V. Gkoutos, 2011), and the main theme was to keep the pages very clean and streamlined. Google has one of the simplest approaches for any website, with only a single form input on its main page. This means that anybody of any ability with computers or the internet, and learn to operate it almost instantly. The results are also displayed in a very consistent manner, with no unnecessary clutter around the edges which focuses the user's attention to the relevant information. Google is also provides a mobile view, which means it works on varying devices, and is tailored for use on mobile view with the reduction of needless CSS and the scaling down of images. This helps to optimises performance on less powerful devices increasing the target audience. In the screenshots below, it is easy to see how the responsive design differs to a desktop version, especially on the web pages which have a table of results.

The project aims to achieve these desired attributes and the first step was producing a CSS template which would look professional whilst retaining full functionality and remain very lightweight. The simple layout adds to the Web Site’s philosophy of keeping everything simple and streamlined, and this allows the user to concentrate on the work they require with no distractions. All results are produced in a very similar manner across the pages which increase the feeling of consistency and help the user navigate their way around. The results tables are also kept very simple, presenting all data to the user without compromise. The tables also have a ‘hover over’ function for each row, which highlights the whole row when a user moves their mouse cursor over it, and this means that it is very easy to scan across and see all information related to a certain disease. There has also been care taken when adding libraries to the web pages, to ensure that there is no unnecessary data being loaded into the page.

From PhenomeNet the project has picked up a lot of subtle additions which make the experience very pleasant for the user, such as having a link to the disease next to the result. As well as this the project has included the segregation of results depending on which genome they are form, e.g. human, rat, mouse, and provides a very quick and easy way to navigate to and from these results despite the results page often being very large. The Project also mimics PhenomeNet by having the ability to start a new query on every page, enabling the user to never need to waste time going back to the start, as well as helping them if they come to a dead end, in terms of no search results.


PhenomeRDF

HOME TREE HELP

DATA CONTACT

Phenotype Tree

Click the links to expand tree, and click search to search for related diseases

Reset Tree

- pigmentation phenotype
- tumorigenesis
- normal phenotype
- no phenotypic analysis
  - End of branch
- nervous system phenotype
  - abnormal nervous system morphology
    - abnormal nervous system tract morphology
      - abnormal brain commissure morphology
        - abnormal dorsal telencephalic commissure morphology

Figure 10. Mobile view of tree.html

Additionally the Web Site needed to be similar in design to PhenomeNet so that users can be comfortable when switching between the two web sites and this will mean that they have a much better experience.

One design aspect which is not taken from either of these two web pages is the responsive design of the website. This is achieved by using CSS rules, to determine how big the device's screen is, and serve the layout appropriately. It means that the page scales itself appropriately depending on the resolution of the screen to provide optimal viewing experience for the user without restricting content. For example on a laptop the layout's width is shrunk to minimize horizontal scrolling, and on a smart phone the layout varies dependent on whether the phone is held landscape or portrait.

All of these factors result in a website which is minimalistic in its approach without compromising content or functionality. The web site is uncompromising on any device ranging from a desktop computer to a smartphone allowing a user to receive full functionality and a fast service regardless of their hardware.

In terms of technical ability among users the project needed to ensure that it could be used by people with even the most basic of computer skills, whilst at the same time, it needs to be made in such a way that a skilled user does not find it frustrating. This meant that the process needed to be very self-explanatory, resisting the need for time consuming tutorials which would frustrate skilled users and add no benefit to the experience. As a result of this, it was imperative to keep the search field as simple as possible, and to only require the user to enter a keyword(s). The search field allows the user to enter either a disease name, gene name or genotype name, which neglects the need for multiple forms which could potentially confuse the user. This also means that the user is not required to have any knowledge of the query language, SPARQL, and so the amount of time needed to learn how to use the application is very minimal, and neglects the need for any over the top tutorial. The web site does offer a page with a few example queries should the user feel the need to get help.

In terms of area of expertise, the application would mostly be used by people who have an understanding of Biology or a related field. The application would not be much use to somebody who does not share this expertise, and so the terminology that can be used throughout the website can be aimed at this audience. In addition to this, the terminology used needs to be similar to the PhenomeNet web site, to enable smooth transitions between the two. As a result there is no need for heavy detail on each result explaining about the terms used which would clutter the page and distract from the important data. It would also frustrate Biologists who would know these terms and they may feel patronised if they are constantly receiving explanations about things.

3.5. Scripts to create the RDF data

The parsing scripts were an essential part of the project and their purpose was to transform the data into a meaningful RDF format. The programming language used for this was PHP as it is a natural scripting language and so it is perfectly suited to the task. The scripts have to parse multiple files at once as well as formatting the output into valid RDF files. The files output by the scripts are split into smaller files when they become too large, so that the server can handle them, and to make uploading an easier task. There are two scripts in total, one has the task of parsing the genotype.sql, phenotype.sql, Inferredphenotype.sql and OntologyTerms.sql files, whilst the other parses the much larger Edge.sql file.

Without these scripts there would be no RDF store for the rest of the project to interact with, and there is nowhere else where the data could have been obtained from. Although the structure isn't too important in RDF so long as the triples are correct, it made sense to give the data sensible names which made it easier for the rest of the project, and will make it easier for any outsider who wishes to use the data to understand it quickly.

3.6. RDF Schema

During the creation of the parsing scripts, it was found that it would be good practice to create a schema for the RDF data so that the scripts had a something to aim for so that it could be determined if they were working properly. As a result a schema was devised for the RDF to follow, which can be viewed in Appendix I, and this was the basis for the final schema which was developed. It is shown in full in Appendix J and is explained below:

Note: phenomeNet shortened to phe: for description purposes.

phe:genotype.

All things described by this class are called genotypes and are an instance of phe:genotype.

rdf:about.

This is used to refer to the globally known identifier. For example "http://phenomebrowser.org/phenomenet/OMIM_187500"

phe:has_url.

phe:has_url is an instance of phe:genotype that is used to provide a URL of a resource.

A triple of the form:

G phe:has_url U

This states that G has a URL U.

phe:has_name.

phe:has_name is an instance of its class that provides a human readable name to the resource.

A triple of the form:

C phe:has_name N

This states that C has a human readable name N.

phe:has_phenotype.

phe:has_phenotype is an instance of phe:genotype that provides an identifier in the form of a URL.

A triple of the form:

G phe:has_phenotype Ph

This states that G has a phenotype with the identifier Ph.

phe:has_inferred_phenotype.

phe:has_inferred_phenotype is an instance of phe:genotype that provides an identifier in the form of a URL.

A triple of the form:

G phe:has_inferred_phenotype Ip

This states that G has a phenotype with the identifier Ip.

phe:has_edge.

phe:has_edge is an instance of phe:genotype that states all instances of it are a sub class of phe:genotype.

phe:Edge.

All things described by this class are called Edges and are an instance of phe.Edge.

phe:node.

phe:node is an instance of phe:Edge that provides an identifier in the form of a URL.

phe:has_value.

phe:has_value is an instance of phe:Edge that provides a human readable representation of the similarity value.

phe:Ont.

All things described by this class are called are an instance of phe.Ont.

4. Coding Implementation and Problems

4.1. First steps and CSS

The first piece of code written for the Project was the CSS file and the simpler pages such as Index.html and Contact.html. The original CSS was obtained from OS-Templates, (Chris Biron, 2012) and has been modified and edited to suit the project’s needs. Using the simpler pages allowed a good basis to ensure that the general aesthetics of the web site were pleasing, before they became too complex to change easily which also allowed for the responsive design aspects to be introduced and perfected.

As the site grew in size and complexity, the simple nature of the CSS meant that it only needed to be tweaked at two points. The first was when the web site needed to style the tables and the second was when the Tree Structure was created. In both instances it was quite easy to accommodate in the CSS, and due to the simple nature of the design there wasn't any complex methods needed to be written. The only major addition to the CSS was the use of images to create the branches on the Tree Structure.

4.2. Creation of parsing scripts

Note: For more information on all PHP methods mentioned below. Visit the PHP Manual online (PHP, 2013)

As discussed in the design section, the parse scripts were an essential part of the project, and took a long time to complete to a satisfactory level. All code was written in PHP for these scripts, and all methods referred to are from the PHP Library. The first step was to create a script which parsed genotype.sql, this was done via the fopen( ) method, and to run code for every line of the file by using !feof( ) . Each line in the file is then split into an array using the fgets( ) and explode( ) method. Because the genotype.sql file has been formatted in such a way that each piece of information on a line is split up using 'tab', the explode( ) method uses this as it's delimiter to split the line into variables inside of an array. Inside of the While Loop, the same process is done to open and parse the file phenotype.sql, and the variables on each line are obtained in the same fashion as above. For each line in the phenotypes file, it compares two variables, which are the disease ID in both files, to see if any match, and if they do it prints a result.

At this stage the script created was capable of matching ID's across the two files, and so it was possible to use very similar code across more files to achieve the same result. However, upon adding more code the script it became very clear that it was becoming very slow due to the amount of time taken to loop through each file and compare variable ID's. It was going to be very time consuming to run a script which took so long to run, and so it needed to be edited to run faster. The solution was quite simple, and perhaps an oversight when initially breaking down the problem. The script didn't need to compare ID's from each file at all, and in fact it just needed to parse the first line of every file and format the results, and then move onto the second line in each file, this meant the script was much faster at parsing the files.

There was then an issue with memory when running the script on the server, as the script was only writing to the file after everything had been processed. The fix was to simple write to the file as it happened, which meant changing from echo( ) to fwrite( ). Also the files being read were closed after each line had been read, and then reopened when it was needed again, to make sure that this didn't cause a delay on the script. This was done using fclose( ), as well as adding a simple counter which told the script which line it needed to access. Even the smallest increase in speed could mean a huge amount due to the size of the data being parsed.
The next problem with the implementation came with parsing the Edge.sql file. This wasn't due to difficulty in actually performing the parsing of it, but it was a result of how large the output file became and so it needed to be handled on its own to make things a lot more straightforward. Due to this, the code for parsing the Edge.sql file was removed from the original script and placed in a script of its own. The script itself differs from the other, as it needs to constantly calculate the size of the file whilst it's being created. It does this using the filesize( ) method and when the file gets too big it starts writing to a new file. This code was straightforward but there was problems getting it to work as the script was caching the size of the file and not updating it. To resolve this problem the script needed to call the method clearstatcache( ), which ensures that the cache for the filesize method is cleared so that it is counted properly.

In addition to these methods used, there are also some more that are important in the script. Firstly str_replace( ) manipulates the string by taking away the colon and replacing it with an underscore, this is needed because otherwise the colon creates problems when it is inserted into URI's. Next the method trim( ) ensures that the variable has had new line indicators, whitespace and the like removed. The need for this became apparent when code that had been written on a Windows Machine was viewed on a Mac, and the text editor, VI, showed the rogue values. The final method needed was htmlentities( ), as it quickly became clear that many of the names in the data set were unusual, and often contained greater than or less than symbol or other values which interfered with the RDF format, and so this method turns these symbol's into ASCII values.

The final result was two scripts which interact with various files, and output the results into valid RDF format very efficiently, as well as being small in size only 6kb and 2kb. The full and complete files can be seen in the Appendix of this document. The script which parses the Edge file is located in Appendix O, and for the larger script which does everything else it is located in Appendix P and continued in Appendix Q.

4.2.1. Maintenance

The nature of the data being used means that it is very likely to be updated and changed on a regular basis and because of this the scripts needed to be designed with the future in mind. The scripts that have been created are capable of handling any updated version of current data providing that the format stays the same. Any new files will need to be catered for, but will most likely use similar code as to what already exists in the parse script, so it would not be a complex procedure. This procedure would have to be done manually at first to ensure that the updated data had not changed format, and if it hadn't the scripts could be ran again to generate the new RDF which would then be uploaded to a new graph on Virtuoso. This presents an advantage because if it was found that the new data was incorrect, then the queries being run could just switch to querying the older graph version.

4.3. Query creation.

To implement the correct queries the project took full advantage of Virtuoso's ability to run queries. This meant that there could be a focus on making sure the query was correct, without having to worry about whether other aspects, such as the connection via the web page was working correctly.

The first SPARQL queries ran was done using the command line interface for virtuoso, using the following command:
*sparql select * from <http://biocrunch.dcs.aber.ac.uk:8890/temp/> where {?s ?p ?o};*

As the data hadn't been fully created at this point, there was some test RDF data created so that the queries could be ran against it. In the command above, the query is asking for all triples from the temp folder on the biocrunch sever, which meets the parameters. The question mark before the letter in SPARQL indicates that it is a variable, and so the parameters in this query are asking for any triple that has a Subject, Predicate and an Object, which essentially means every triple.

After this, it was found to be easier to use the SPARQL query engine on the Virtuoso application rather than the command line. This allowed for better debugging of queries that were invalid, and a better display of the data returned from the query. After becoming more accustomed with SPARQL queries and how they worked, the next stage was to create the queries that would be used on the website. This led to understanding what each page would require and using that knowledge to come up with a suitable solution. For example, diseases.php would need a query which searched for all diseases which had a name that matched the one required. At this stage the project had some level of working functionality and was able to query data from Virtuoso using the created queries. However,, for the purpose of this project they needed to be put into a web page, so that they could query the data set from there. This interaction is done using the RAP RDF API library, discussed in more detail later on in this section.

The next stage was to learn how the queries differ in syntax when they are using the library and how to output the results found from the query. This led to the creation of a PHP page which performed a simple query:

```
select *

FROM <http://biocrunch.dcs.aber.ac.uk:8890/DAV/complete>

where {    ?s ?p ?o

           }";
```

As shown this is very similar to the query that was ran on the command line in the early stages of development. The difference is that this query cannot occur on it's own and uses PHP to enable the execution of it. The first step required is to define and include the library in the page, which uses the define( ) and include( ) methods. The endpoint is assigned to a variable and is stated using the getSPARQLClient( ) function from the RAP API RDF library. The whole query is also assigned to a variable, and it's syntax only differs by needing to be inside quotation marks as is standard in PHP. To execute the query, a new ClientQuery( ) method is created and passed to the SPARQL client.

The printing of results requires a foreach( ) loop which goes through all of the results, and each variable used in the query, e.g "?s", is then assigned to a PHP variable so that it can be printed to the page.

Due to the way the data has been stored in the RDF data store to enable a better sanitisation of results there are a number of PHP methods called. Preg_match( ) uses regular expressions, and is needed to get the ID of phenotypes and genotypes as these are sent as full URI's, and the only part needed to be displayed is the Unique number, e.g. OMIM_0000001, which is the last part of the URI. Also needed is str_replace( ) which is used to manipulate the Unique ID's further, as the start of them e.g. OMIM is not needed when it is passed to the external website which specifically deals with OMIM diseases.

After this part was working, the next stage was to implement the use of PHP variables within the query. The first method tried was to put the query inside the sprintf( ) function, which means part of the query looks like this:

SELECT DISTINCT ?s

where { <%s> ?o ?p . }', $searchQuery) ;

A problem with this was that the variable was often going to be only one part of the string which it searched, e.g. Alzheimer, and the disease names could be Alzheimer disease, and using this method they would not match. So another method had to be devised in order to find a solution to this problem. The solution came by using the Filter methods in SPARQL, and although this is slower it provides much better results.

FILTER regex(?dis, '$searchQuery', 'I')

What this does, is assign the PHP variable $searchQuery, to the triple value ?dis. So if they $searchQuery value was 'OMIM_605055', and the query asked for all diseases which have a phenotype. This would translate as, all diseases which have phenotypes limited to all diseases which have the ID OMIM_605055. This is slower as it returns all results before filtering the output of the query.

The final adjustment to the queries increased the speed by a small margin. Instead of using SPARQL's Filter option, it was discovered that variables from PHP could in fact by put straight into the query. This meant the filtering was done automatically and at the start of the execution, so time isn’t wasted returning all of the results. The speed increase was very noticeable, for instance the results for the page ontologyterms.php was brought down from around 15 seconds, to well under a second. The only page it could not be implemented was on diseases.php as this requires regular expressions to match singular words with whole phrases. The new code is shown below:

$searchQuery phe:has_inferred_phenotype ?infpheno

An example of diseases.php is available in Appendix M of this document. All other queries used in the application are built in a very similar way.

4.4. Creating the Tree Structure

The first steps in creating the tree structure were to understand which technologies would be required and which parts would interact with each other. It was obvious that there would be a need for AJAX to create the dynamic content required to produce a tree structure, but additionally all of the processing couldn't be done on the client side, so there needed to be a script which ran on the server. The script was the first challenge, and to be able to focus solely on this, the AJAX parts of the feature were left out in the initial stages of the implementation. This meant the script could be created without having to worry about combining it with the other technologies, as a result the script is written solely in PHP.

To begin with the script was designed to parse the mammalian_phenotype.obo file, and collect the first level of child elements. Knowing that the Root of the ontology was MP:0000001, the script can use this to search for all entries which have a relationship to it. The script then converts the whole contents of mammalian_phenotype.obo into a string using the method file_get_contents( ), and is immediately broken into an array using the explode( ) method, which uses "[Term]" as a delimiter, as it is used in mammalian_phenotype.obo to declare the start of a new entry. The script then prints some HTML to the page, and uses the variable to assign an ID in the <div> tag which is essential for the jQuery on tree.html to work.

The script searches through the file and finds all entries which have a parent ID which matches the original phenotype ID given to the script. The script has to use the PHP function 'preg_match' to find the line in each entry which begin with "is_a", as this indicates what the parent phenotype is and then uses explode to split the term up. Two more functions, str_replace and trim() are used before it is ready to be compared with the original variable. The script then prints out all results that match with HTML mark up, and this is what the jQuery uses to produce the tree. At this stage the script is functional, and performs all the task required of it. Each time a link is clicked it can use the ID of the clicked link, and send the variable via GET to itself, and the script then runs using a different parent ID, and gets there children. The next stage was running the script via AJAX, so that it would print the HTML dynamically, rather than to a new page each time. The full code for treescript.php can be seen in Appendix N of this document.

This led to the creation of tree.html, which calls the script when it needs it. The page originally used Javascript, but due to ease of use the language used was changed to jQuery which offered much better support for the task to be completed. The end result was a very quick to load, and responsive tree, which allows a user to quickly navigate their way around it.

Note: For more information on all jQuery methods mentioned below. Visit the jQuery website which provides documentation. (jQuery, 2013)

When the page loads, the jQuery performs a function which gets the results of treescript.php, and displays it to the user. When a link is clicked, jQuery performs a different function, which gets the ID of the link and passes this using POST to treescript.php. The results from treescript.php are then displayed dynamically below the link that had originally been clicked, jQuery does this by using the ID given to the div tag by treescript.php, to create the effect of a tree structure. If the same link is clicked twice, a jQuery check to see if this link ID has already been clicked, and if it has it removes the child elements, creating the effect of the tree closing. This is done using an array, which stores the various ID's when they are clicked, and when the a link is clicked in future, it compares it with the strings in the array to see if it has been clicked previously, and if it has, it removes the child elements, as well as removing the ID from array. This gives the impression that the branch is open or closed depending on whether the variable is stored in the array or not. At any time the user can click the “[Search]” link next to a phenotype, and this will send the phenotypes ID as a variable to phenotypes.php via the GET method in PHP. Additionally the user can choose to reset the tree by clicking the 'Reset Link' which will return the tree to its first stage. The tree structure would not be possible without both tree.html and treescript.php, as they work together to provide the final result.

There were a few problems during the creation of the Tree Structure, and most were due to the wrong method being called in jQuery. For instance the subtle difference between .empty( ) and .delete( ) was key, as the former will remove all child elements whereas delete removes everything, so it was a question of ensuring the correct method had been used. Additionally there was a need to give the div tag's an ID so that they could be identified by the jQuery which needs an element to focus on when it is performing the .after( ) or empty( ) function. Without the ID it would affect all occurrences of whichever tag is stated. The ID is unique and so only the elements with that ID will be manipulated. The jQuery algorithms of tree.html can be seen in Appendix R of this document.

One advantage of this approach is that if an updated mammalian_phenotype.obo file becomes available, providing it is in the same format, it can simply replace the existing one, and the tree will immediately have all of the new and updated data.

4.5. Non Coding Implementations and Problems

4.5.1. Initial Stages

During the first stages of the project, it was very difficult to gauge an understanding of what the final product would look like. This was mainly due to a lack of knowledge about the project itself, and the technologies involved. As a result a lot of research was needed to be able to fully comprehend what the objective was, as well as understanding the technologies that would be relevant to the project, and this was the first step that the project took.

After the background research, a few Mock Ups of what the finished application should look like on a Web Site were created. These were done using a tool called Balsamiq Mockups, and enabled a fast way to provide very rough drawings. These were done very near the start of the project but the initial design have a similarity to the final website design, which is quite pleasing. These can be viewed in Appendix L.

However, it was quickly discovered that it was very difficult to design a layout to present data, when the data wasn't fully understood. This meant that more time was spent learning about the data the project would be dealing with, and the relationships between them. Such an in depth understanding made much of the project a lot easier to implement, especially things like creating the queries as it was clear to see which pieces of data were required. Despite the initial delay it is obvious that the extra time taken at the beginning saved time later on in the project.

4.5.2. Data Management System.

The next step in the project was to set up the data management system. The main choice came between 4Store and Virtuoso (Virtuoso, 2013), both of which are very well established and powerful applications. This was an important choice to make due to the size of data the project would be accessing, even a marginal increase in resource usage or speed could have a knock on effect to the application. In a benchmark test in 2011, (Chris Bizer and Andreas Schultz, 2011) Virtuoso outperforms its competitors in many tests. As a result, coupled with the huge array of features that it offer it makes it an obvious choice for the project.

Virtuoso is a very powerful application in its own right, allowing for command line, or browser interaction. It incorporates a vast amount of different aspects of common database functions, such as Content management and even Mail storage, as well as other abilities, into one place thus the “universal” part of the name. The main features related to the project, is its capabilities with being able to hold an RDF store, as well as being compatible with the SPARQL query language. Using the built in SPARQL query tool, users can perform simple to complex queries on the data that has been uploaded. It was installed via the command line on Unix based systems which included the server, and via an installer programme on windows.

The first problem that was discovered whilst using Virtuoso, was the command line interface. Virtuoso allows users to upload data via command line, which makes it perfect for uploading files from a server without the need to copy them over to a local machine first. However, as default setting Virtuoso only allows a couple of directories to be used, and these include the /tmp on Linux machines and the directory in which Virtuoso is installed, which meant that it was not possible to upload data from the server.

Due to the excellent documentation provided on the Virtuoso web pages, and the Open Source community, a solution was quickly discovered which allowed the user to choose more directories. This didn't cause too much disruption to the project, and in retrospect it was merely a learning curve of employing a new application. There were a few more cases of similarly trivial issues that were addressed resulting in facilitating a simple upload mechanism and query testing.

A bigger issue that was encountered using Virtuoso again came through the command line interface, and this time it was a problem with uploading the data. This is discussed in detail in the next section as it also includes a number of other factors which are yet to be explained.

4.5.3. Creation and uploading of the RDF

As mentioned before, understanding the data was a very important part to the project, an issue that became more prominent when it came to parsing the data given into RDF format. Before creating a script to parse the data, there needed to be an understanding of how the data will look after being parsed. Again this was a case of understanding the objective before being able to write the method which produces the result. This meant that a schema needed to be created, which would help to describe the data, as well as provide a skeleton template which the parse scripts could use to format the RDF properly. The schema needed to be completely correct, as any small error would be replicated thousands of times due to the amount of data being parsed. Consequently this was not the case, but as a precaution only the first one hundred lines from each file being parsed were used so the errors were easy to fix and as a result not very time consuming. The biggest, and most time costly errors came from when the whole data set was put together and uploaded to the Virtuoso data store.

Firstly, the creation of the full data set due to its size, took many hours, for the largest file this was around eight hours, and would create a file that was over five times the size, mainly due to the mark up included in the file as opposed to the stripped nature of the original data. This meant that any time there was a problem with the server or if there was an error with the data created, the whole process would have to be restarted.

At first there was an issue with the file becoming too large to exist on the server as a single file, as the server didn't have sufficient memory to create it. As a result it had to be parsed into a number of smaller files. This then created the added step of having to knit the files back together once they had been created. This wasn't too difficult in terms of procedure, but it was very time consuming waiting for the process to run. Additionally, because the files were split at intervals dependent on their size there were many cases of RDF statements being cut in half. This meant having to manually fix all the problems, using command line editors such as VI, which wasn't too difficult but again took a long time. Another factor with manually editing the RDF, was that it is a lot more prone to errors by human typing, than when it was being parsed by a script, and this caused problems when uploading the RDF.

As mentioned the only way to upload the RDF was through the command line interface provided by Virtuoso. The method of doing so was very straightforward and required just one command, but the issue was regarding the lack of feedback during the upload. While uploading files that were often well over 20 GB, there was no way of knowing how long the process was expected to take.

This was probably the worst aspect of Virtuoso that was found during the project, and it hindered the project massively, especially when there was an error in the RDF. When this occurred the command line interface gave no indication that there was a problem and appeared to be processing the upload as normal. However,, it would eventually stop and send an error back to the console about what was wrong, this feedback itself was helpful but the time it took to produce this feedback was often lengthy. For example whilst uploading a file that should of taken about eight hours, the console didn’t return an error for three days, and still hadn't before it was terminated manually. On other occasions it would take longer than expected, only to return an error message regarding Line 1 of the RDF file. This process was very time consuming and every time there was an error the whole process had to be restarted as there is no way of determining how much data, if any, had been uploaded, and there was also no way of starting an upload from where the last one failed.

The whole process of getting all of the data loaded into the Virtuoso RDF store took place over a period of about two weeks and as a result further delayed many other aspects of the project taking place. This is because for some functions, namely the comparison of diseases, it required a very complete data set for it to be tested properly. Even before the queries got that complex, it was still difficult to analyse if queries were correct if they were returning no results.

4.5.4. Libraries

The project had a heavy reliance on finding a library which allowed for the interaction between PHP and Virtuoso via SPARQL. After researching a few possibilities the only way of knowing for sure which would be suitable was by implementing the code and testing it. This was one of the first major steps in the project, and so it was important to pick a library that was easy to work with, and provided a lot of support.

The first library used was called LibRDF. One of the main issues with this library was the lack of documentation, making it very difficult to understand the abilities the library had, and whether it suited the project's needs. Despite these issues, it was decided that it was worthwhile to try and attempt to use this library, however after spending a few days trying to configure code to our needs, it became increasingly frustrating to have no documentation to look at. In hindsight it was very possible that this library would have been suitable but the time constraints on the project meant that we could not afford to waste time chasing red herrings and so it was decided that a new direction would be needed to take the project forward.

This led to the discovery of a different library called RAP RDF API 0.96 (Daniel Westphal and Chris Bizer, 2004). This hugely contrasted to LibRDF, as it had a huge amount of documentation and example code, which made the task of querying the RDF data very straightforward. To make sure that the library was suitable for the project, some RDF examples was constructed, and then some very basic queries were tried out to ensure that it worked. Using the documentation supplied, we were able to perform queries locally as well as on the data stored on the server.

The results were very pleasing and we were satisfied with how smooth the queries ran, as well as the flexibility it gave when printing out results. This was important as the results needed to be harnessed and displayed in a controlled way to enable the fulfilment of the projects objectives. Again, the time constraints of the project were factored into the decision, and it was decided that it wouldn't be logical to search for other possible solutions after finding one that did everything the project needed. It was at this point that we felt safe that this was the correct choice for moving the project forward.

5. Testing

5.1. Overall Approach to Testing

Following the Feature Driven Development methodology, testing was performed at the end of the 'Build by Feature' stage. This would involve a code review, which helps to fix mistakes that may have crept into development during the previous 4 stages. Despite this, FDD does not have any more guidelines on how to test an application and it is left to interpretation, and as a result the testing can be altered to fit any project, as there is the choice of any testing technique.

Many testing techniques require more than one person to perform, such as pair programming, and so they are not suitable for this project. The testing environment used was quite simple as this was not an object orientated application, so it didn't allow for unit testing. Therefore PHP was installed locally using an application called EasyPHP which stores all error's in an error log, as well as turning all possible warnings and error messages were on to provide maximum feedback on any problems with the code.

5.2. Feature Testing

During the creation and upon completion of each feature a series of testing techniques were applied. As mentioned above FDD is quite flexible and so the techniques which most suited the project were chosen and applied. This meant that techniques could be tailored to the individual nature of the project rather than being rigid and ill fitting. Below is an explanation of how each chosen technique suited the project along with examples of where it was used.

Print Debugging.

This involved watching the code run live by using print statements and monitoring its progress. This task employed techniques such as 'echoing' variables to the page, so it was it was possible to check whether the code was performing as was expected and in the correct order. This was essential during the development of the PHP pages which ran queries, e.g. diseases.php, as it provided visible feedback which could then be used to gain an understanding of a situation. For example, during the creation of one page, via printing out the variables during the running of the script it was apparent that only one of the variables was being printed out, and the other was failing to do so. This was realised by, and it was soon discovered that the problem was a variable name not being the same as the one it should be. This was a common problem during development, and being able to use this simple debugging technique meant that it could be identified and fixed very quickly.

Remote debugging.

Remote debugging is where the code is ran on a different machine or server to the one it is being tested on. As a result it is possible to see how the code behaves in the different environment, and to see if anything changes as a result. In this project, to perform remote debugging, the code was transferred from the local environment to the server which would host it in the same fashion. This exposed any problems which had been hidden during the testing in the local environment. One of these problems was the use of local files, such as libraries, which were not present on the server. Another issue was the use of a local SPARQL endpoint, which had been implemented in the early stages and had simple been forgotten about, this was quickly identified due to the helpful error messages presented by the server.

Isolated debugging

This method is reliant on the identification of a section of bad code, which can be taken out of its current position and tested on its own. This usually means assigning local variables, or hard coding them, as opposed to relying on variables sent from other areas which may be causing the problem themselves. This technique was used during the creation of the parsing scripts, which suffered problems with the parsing of the edge.sql file as it was more complex in comparison to the other parts of the script. By removing the code from the script, it was firstly evident that the rest of the script performed properly which wasn't possible whilst the broken code was still there. Secondly it meant it was much easier to fix the isolated code as it ran much faster as well as being uninfluenced by external factors. After the code was fixed, it was decided that it would serve better to leave it outside of the main script as it wasn't quite suited to be mixed with other functions as its own was complex enough. Another aspect of this debugging technique was the use of hard coded variables, which eliminated the reliance on other functions of the system, thus isolating the section from the rest of the code.

Post Mortem Debugging

Usually this refers to debugging code after it has crashed or had a fatal error, but in this project it was used to debug code after it had performed an operation regardless of whether it had crashed or not. The main case where this was used was upon the creation of the parsing scripts. When the scripts were ran, they produced no errors, and everything appeared in order, however upon inspecting the RDF which was created using the RDF validator online it showed that the RDF was invalid. This meant that there was a problem with the way the script was outputting the data it was parsing and by using this output the code could be amended and shaped until the output was correct.

The same technique was also used after the creation of the query pages such as diseases.php, as the output could be analysed to see if it matched the expected input, and if it didn't the code could be changed until it was operating as expected.

Code Inspection

When it could be decided that the code was performing all of its duties properly, it was then subject to a number of other testing procedures which assessed the logic and quality of the written code. As mentioned Feature Driven Development is usually used within large project groups, and as a result many of the testing techniques are not possible, or have to be adapted to suit this project. The first process which had to be changed was the Code Review as it requires a meeting amongst the project members as well as assigning different "inspection roles". As this was not applicable to the project, the code was set aside for a period of time, so that it could be approached from a fresh perspective, and when reviewing the code comments were written alongside the code to describe each stage of the various methods. This is sometimes referred to as "Rubber Duck Debugging", whereby a programmer talks aloud about the code they have wrote, comparing what it does to what it should do, and the difference becomes obvious. This approach is not as productive as it would be to have a peer review, but given the specific circumstances of the project it provides a very useful feather that helped to optimise large parts of code. An example of this was when the queries had been created, it became apparent that the query was not selecting a specific graph from the Data Store and as a result was pulling all of the data instead, which meant that the result were producing duplicates as well as incorrect data. This was easily fixed, but without the review it would never have been picked up and the application would have suffered a performance loss.

5.3. Acceptance Testing

The next stage in testing each feature was to perform acceptance testing, this test to see if the requirements of each feature has been fulfilled or not. This uses the feature list which was built during the second stage of the process model, to provide the 'yardstick' for which the success of failure can be measured against. Acceptance testing itself can be split up into further sub categories, however during this project only User Acceptance Testing and Regulation Acceptance Testing were used. Acceptance testing on this project was done using the Black Box testing technique, which doesn't worry about functionality, but is more concerned with the output in comparison to the expected output. The results of the acceptance testing on the final application can be seen in Appendix F.

5.4. Integration Testing

Integration testing was performed after two or more parts of the application were brought together, and at this point the functionality and performance of the application is assessed. This is a very important aspect of the project as it involved many different technologies working together and so it needed to be determined whether they were operating as expected or not. For a feature to be determined as successfully integrated with another, the flow of data between the two parts must be inspected. The results of this are shown in Appendix G.

5.5. User Testing.

During this stage of testing a pool of users with varying background and computer skills were given access to the application, and asked to carry out the two scenarios below.

Scenario 1:

Task: Search for a disease and explore the diseases related to the choice, before viewing the associated phenotypes. Appendix S shows this step by step process.

Expected process:

The User submits a keyword and is taken to diseases.php where they are shown a table of diseases which match their query. The user then chooses to explore a disease to find related results which takes the User to edge.php. Here the User is given another table of results, where they choose to view the phenotypes of a disease and this takes them to ontologyterms.php. Here the User is shown two tables, one with phenotypes that are directly related to the disease selected and one with phenotypes that are inferred. If the User wishes they can select a phenotype and be taken to an external site with more information.

Actual process of Users: (This is unedited feedback from users of the system and may contain grammatical errors.)

User 1.
Occupation: Studying field of Business Information Technology

Comments:

“Searching a disease is really easy. I entered 'cancer', I was then directed to a page that displayed diseases which matched the query. This then gave the option to explore all related diseases to the results. I was then able to click phenotypes on whichever disease to display the related phenotypes.

Overall a quick and simple step by step process.”

User 2.
Occupation: Biologist
Comments:
“Clicked in the search box on the home page. Entered 'Lung', pressed search to display listings. Scrolled and had a look at results, clicked explore on 'SMALL CELL CANCER OF THE LUNG (OMIM_182280)' this brought up edge names. Clicked on phenotype link for 'ZINC FINGER- AND BTB DOMAIN-CONTAINING PROTEIN 16 (OMIM_176797)' to produce list of phenotypes directly associated and inferred. Clicked on HP:0000006 phenotype link to take me to https://www.ebi.ac.uk/ontology-lookup/?termId=HP:0000006, the ontology lookup service.

The site was well laid out and really easy to use with data easily found and displayed appropriately. The explore page did take quite a while to load however I guess this is down to the number of results produced and large query size, and therefore should be expected.”

User 3.

Occupation: Biologist.

Comments:

“Entered the keyword “Alzheimer” and was presented with a list of results. I wanted to see what phenotypes were related with the first result so I clicked phenotypes and was given a list of results. I then clicked back in the browser and chose to explore the first result. I was given a list of related diseases.”

User 4.

Occupation: English Student.

Comments:

“I used my mobile to search for heart disease, and was given a table of results. I then scrolled down the page and picked a result at random and clicked explore. This then gave me more results. I clicked phenotype on the first result and it gave me a list of what looked like symptoms related to the disease.

The site was really good for mobile use as the tables were nicely presented and it felt very clean.”

User 5.

Occupation: Computer Science Student.

Comments:

“I used my iPad to visit the website, and was presented with a well-designed homepage, which fit nicely on my device. I used the HTML form to search for “liver”, and was taken to a new page after a few seconds from when I pressed enter. The page had a large table full of results from the search. The table had hover over features which made it easy to see which row I was on. I selected explore and was taken to another page again after a short delay. This page displayed a list of results which seemed to be related to the previous page. It was very similarly laid out also. I then clicked phenotypes and was immediately taken to a page which showed me a list of results based on the disease I had chosen. This was also laid out very similar to the previous pages.”

Scenario 2:

Task: Use the Tree Structure to find diseases related to the phenotype of their choice. Appendix T shows this step by step process.

Expected Process:

The page containing the Tree is accessible from every page via a link in the menu which remains the same throughout the website. The User is then presented with the Tree which loads when the page does. The User then selects a branch they are interested in, and this opens up a new set of branches relating to the clicked link. The User repeats this process until all Child branches are exhausted or they decide to click the Search function. Once clicked the User is passed to phenotypes.php where they are given a table of results which show the diseases that are related to the phenotype selected.

Actual Process: (This is unedited feedback from users of the system and may contain grammatical errors.)

User 1.
Occupation: Studying field of Business Information Technology.

Comments:

“The tree is a really effective part of the site giving an easy method for searching through phenotypes. The hierarchical view makes navigation really easy, so much so that anyone could understand how to use it. I used the tree to find a phenotype and then clicked the search icon. I was taken to a page which gave a list of diseases.”

User 2.

Occupation: Biologist.

Comments:

“Clicked nervous system phenotype, abnormal nervous system physiology, abnormal dopamine level, clicked search icon next to 'increased dopamine level' to bring up list of diseases.

The process of being able to visually see the different options available and what levels the different types of phenotypes are related is really good. It’s also nice to be able to eliminate and narrow down the search in this way without having to know an exact name. The search at the end also performed quickly.”

User 3.

Occupation: Biologist.

Comments:
“I navigated through the tree looking for an interesting phenotype, and chose ‘meteroism’. I then clicked the search icon next to the word and was taken to a page which presented me with a list of diseases which were related to the phenotype. The tree is a great way to quickly look for diseases related to a phenotype, but It could be improved by being ordered alphabetically.”

User 4.
Occupation: English Student.

Comments:
“I really liked the tree as it was an enjoyable way of searching through different phenotypes. It worked great on my mobile and was not restricted by the screen size”

User 5.
Occupation: Computer Science Student.
Comments:

“I visited the page with the tree structure and played about with it to begin with. The tree was very responsive and fast to load any branch. The simple design was very good as it didn’t provide any distractions. After a while I clicked one of the search icons and was taken to a table of results, laid out like before. The page loaded very quickly. The whole experience on my iPad was very pleasant”

6. Evaluation

Overall, the project has been a success and I have gained a great deal of knowledge from a lot of different technologies. The overall goal of the project was to produce a web application with an interface that allows a user with very limited computer knowledge to query an RDF store, and this has been achieved. There have been many obstacles throughout the project which have challenged my ability and forced me to do some further research to understand the problem at hand. This has greatly increased my knowledge of not just the technologies, but problem solving in general and the different ways of breaking down a problem and achieving a solution.

During the initial stages of the project I found it difficult to understand the capabilities of all of the different technologies, and as a result it was hard to grasp what the limitations of the project would be. It was known that an application needed to bridge the gap between biologists and the huge data stores they kept, but it was hard to come up with a complete plan which would achieve this. This meant that a lot of time was spent researching the different technologies used which slowed down the project. In hindsight this was the wrong approach to take, because without the knowledge of what was needed to be done, it was hard to research the correct topic. I was able to learn much more about the technologies when I started completing some small pieces of spike work which were related to the project. This allowed the project to move forward as well as increasing my understanding of the various technologies. I can learn from this experience during future projects and approach them in a different manner.

This lack of understanding meant that it was difficult to understand how to achieve the requirements set out, and in some cases what the requirements actually meant. Despite the initial confusion the requirements became clearer as the project progressed, and the final outcome meets all the objectives that were set. In Appendix S there is a step by step guide of the process a user undertakes to enter a query and find its related diseases and phenotypes. In Appendix T, there is a demonstration of a user using the tree structure to find diseases related to a chosen phenotype.

The final presentation of the website is clean and simple which is what was strived for from the outset, as it does not require anything more than the functionality it provides. As seen in the design section, the website created looks professional and the use of responsive design means that it is accessible on smart-phones which increases the number of users capable of accessing it. The lightweight nature of the website also means that it is unlikely to suffer any performance problems on older machines, as well as in areas with poor internet connectivity.

The overall performance of the web application in terms of speed is not as fast as it could be. For instance ontologyterms.php which loads the related phenotypes for a disease provides instant results, which indicates good performance. Whereas edge.php is usually a bit slower, often taking around 5 seconds, this is due to having 8 queries being performed on one page. There is potential to optimise these pages, and if the project was done again, I would ensure that the queries were not all executed at the same time, and are instead executed when the user requests them. For example, the user may only require the related OMIM diseases, but at present they are returned all data categories which slows down the return of results. The delay is not so bad that a user would feel the need to exit the application, but as mentioned there is definitely scope to improve the application in this area.

One problem which will be lessened after the application is made public is the initial delay of getting results from keywords or ID's which had not previously been queried. This is due to caching on the server, and so as the application is used more frequently it will be able to provide results much quicker.

The design of the project served well throughout, and the agile methodology used allowed for the overall model to be changed very frequently which aided the project as it was a constant process of learning and understanding new things. There were a few minor omissions such as not assigning a data type to the similarity value in RDF. This meant that the value could not be filtered as it was treated as a string, and the data files had to be parsed again. This is quite simple to fix, however the time taken to re-upload the data would mean that the deadline for this project would have been missed. This is definitely an aspect which could be improved if the project was to be done again, or even in future if deemed necessary. To cope with this omission, results are limited to the top 200 results which provide a good amount of data for the user as they are usually only concerned with the top results in most cases.

As touched upon above, one of the biggest failings of the project was the process of getting the data which had been parsed, loaded into the data store. Unfortunately there was no other alternative as it would have been a risky strategy to start trying to use an alternative to Virtuoso at such a late stage. Additionally, the problem was only a big issue when dealing with the larger RDF files. For future projects it might be necessary to research a data store which provided a better way to upload data, or which gave more feedback. If this is not possible it would be good to avoid uploading large files, and have them split into smaller parts as this would make the process much quicker and errors would be reported much quicker. Despite this, Virtuoso was still a great tool to use and I think it would be a leading choice in any similar project in the future, as this was the only drawback.

The most pleasing aspect of the project from my point of view was the development of the tree structure. This was a difficult task which brought together PHP, SPARQL, and jQuery to create the dynamic content. It was important to create the tree structure using only the pieces of data needed at any one time, which meant that the whole file couldn't be loaded straight away as this would effect performance. I am very proud of the final result as the tree is very fast and responsive and is fit for purpose. The only minor change I would make would be to sort the list alphabetically, other than this there is a basis to build upon the existing structure and create trees for other ontologys.

A feature I would like to include in the project is a compare function which would allow the user to select multiple diseases or genes using a check-box system and then view them side by side, so that it is easy to see which phenotypes the diseases have in common. I feel this is a very realistic aim, and in my eyes would be the next logical step for the project after some of the other smaller problems have been corrected. Additionally I would also like to offer a quick step by step tutorial for users who require extra assistance. This would most likely involve setting a cookie to see if a user has not completed the tutorial, and ask them to partake in the tutorial. However,, this would need careful planning to not frustrate users who do not need the assistance.

One part of the development process which was not tested enough from my point of view was the RDF Schema. As alluded to earlier on in this document, even the smallest of errors in the parse scripts will have huge implications as they will be repeated thousands of times. In hindsight I would definitely ensure that the schema had been tested thoroughly, not just for syntax errors, but for logic errors too. I would've also liked to carry out some PHP Unit testing, but due to time constraints there was no time remaining to do so as I have not completed any PHP Unit testing before, so a large amount of time would need to be spent learning how to do it. This could also impact the actual testing which was carried out, if it wasn't done properly. However,, I feel that aside from these cases, the testing which was carried out was sufficient as it provided a good assessment of all the features which had been completed and whether they carried out the job required or not.

PHP proved to be a very good choice of programming language, being used heavily throughout the project and is accountable for 91.9% of all coding language used on GitHub according to the project's GitHub repository. The created PHP scripts run very quickly and are very lightweight. As a result I would definitely use PHP again if I was to create a similar project. In contrast to PHP, of which I had a basic understanding, I had no prior knowledge of the Resource Description Framework (RDF) before beginning this project, and the same applies with SPARQL. These technologies were a necessity in the project and would be needed in any future projects with similar aspirations. Additionally, I found SPARQL very pleasing to work with and was able to achieve all objectives using it. One tool that was outlined in the initial stages as being needed was YUI (Yahoo! User Interface), this was going to be required to create the tree structure, but in the end it was not needed as jQuery and CSS performed the job equally as well.

In regards to the target audience, I believe that the project successfully delivers to the targeted user audience as the application requires minimal technical knowledge to operate, which was one of the requirements of the project. Users find it easy to create queries as they only need to type in a keyword or phrase. The application caters for different types of user, it can also quickly assist somebody who knows exactly what they want to search for, and conversely it can help somebody who just wants to browse a certain area.

In terms of the biological audience, I feel the application definitely meets the target audience's needs. The terminology used throughout the application is leaned towards a user with a biological background. As well as this, biologists can use the application to find information that is relevant to their studies and this is the main goal of the project, to bridge the gap between biologists and the data.

To conclude, I believe the project was a success as it has met all the objectives set out in the initial stages. The application provides a tool which biologists can use to query vast amounts of data across various species, to find diseases which are related by their phenotypic value. The project has also provided a very strong foundation for the application to be developed further in the future to include new data as well as new features and functionality.

Appendices

A. Third-Party Code and Libraries

The main library used in the project, as discussed in the implementation section of the document, was RAP RDF API 0.96, which allows the querying of RDF data using SPARQL queries via PHP. Using the tutorial on the website, the "Using the SPARQL Client" part was followed. Using this code enabled the querying of the RDF store located on Virtuoso.

Web Address: http://wifo5-03.informatik.uni-mannheim.de/bizer/rdfapi/tutorial/usingtheSPARQLClient.htm

The code shown in this tutorial is what has been used as a guideline in the application, but it has been changed to suit. Without this code, the querying of RDF using SPARQL in a web application would not be possible, as PHP does not provide this facility.

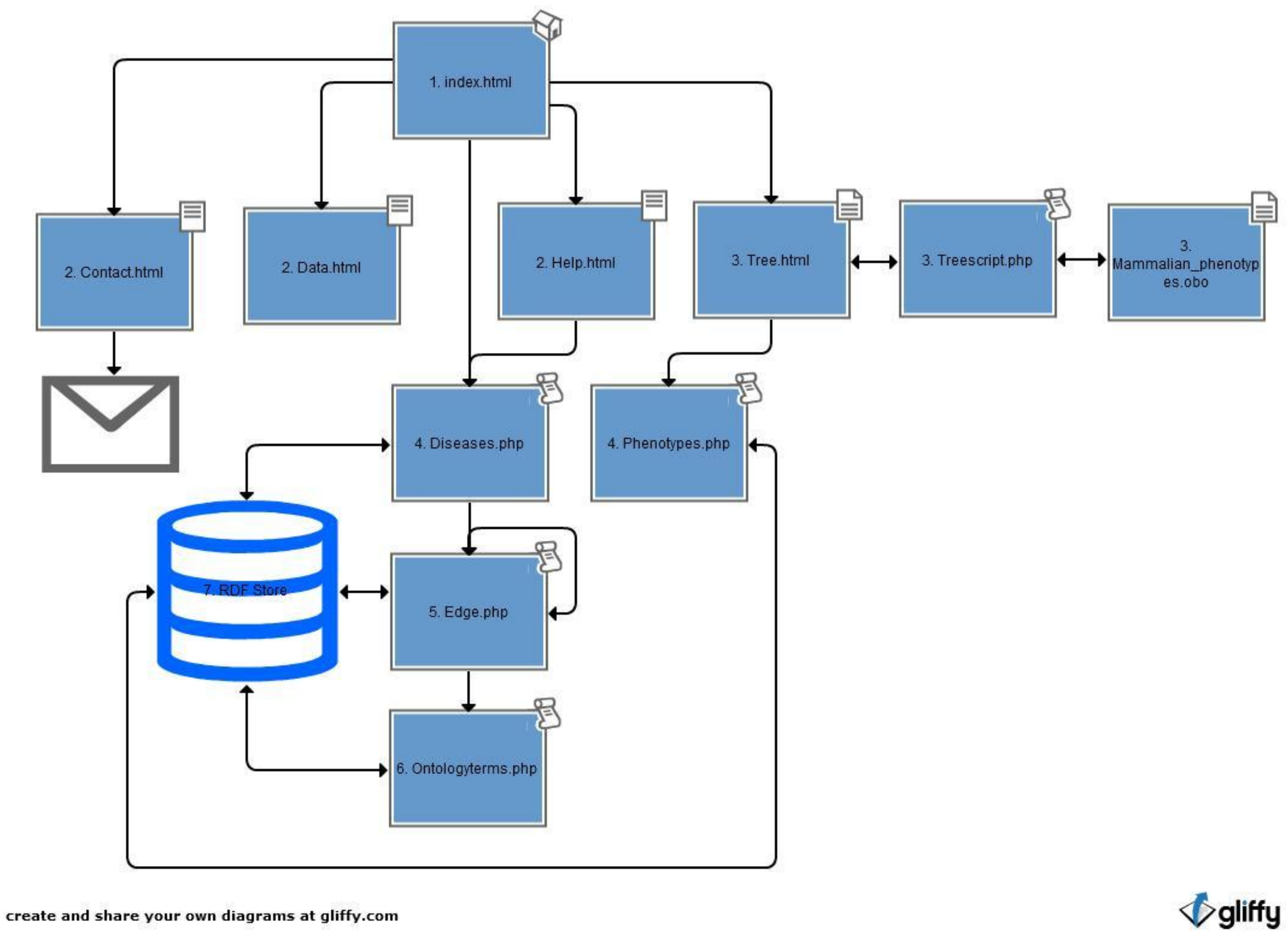
As also mentioned the CSS for the web site is a template which has been edited to fit. Web Address:

http://www.free-css.com/assets/files/free-css-templates/preview/page161/liquid-gem/

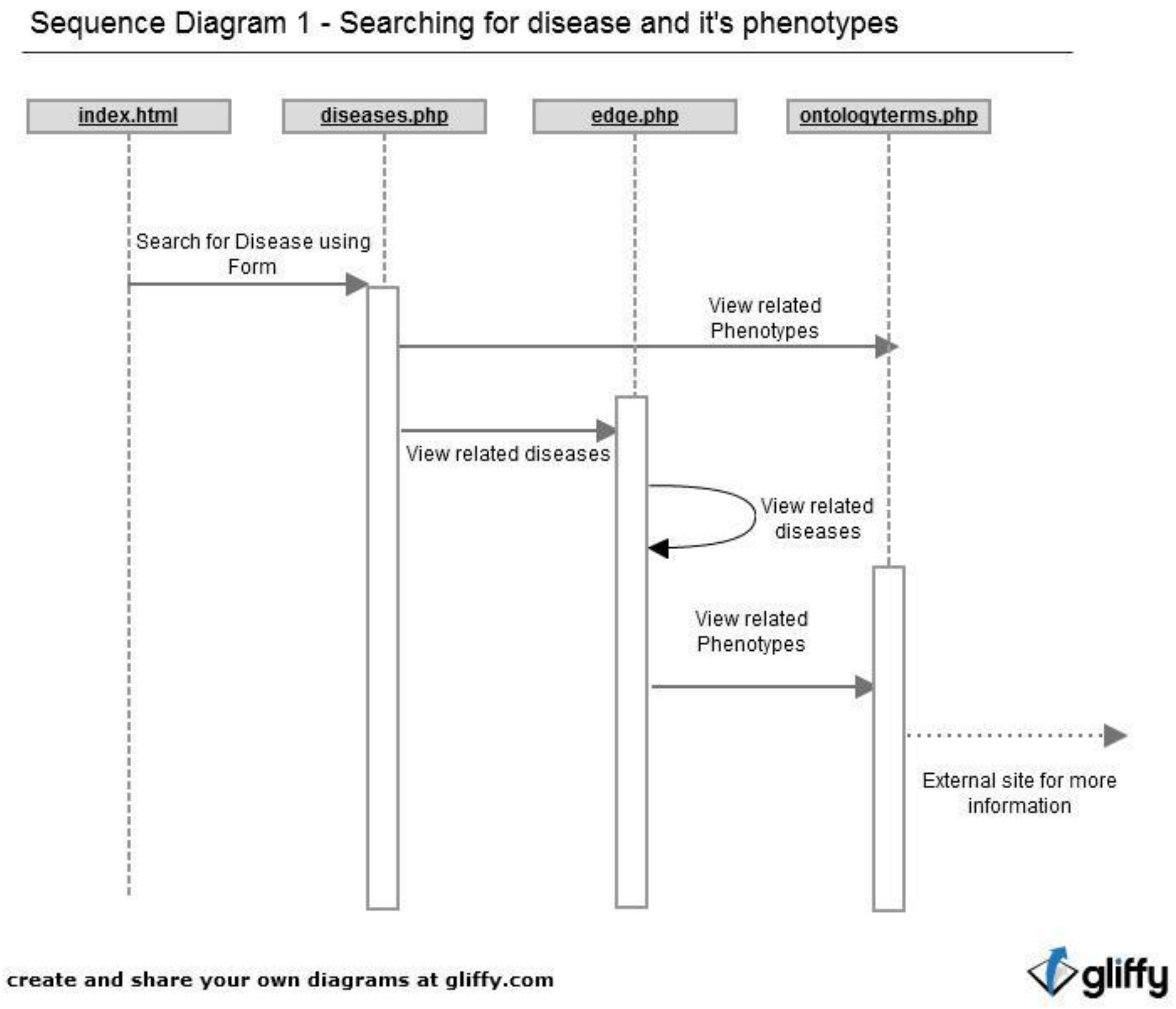
Additionally the CSS for the tables was obtained by following the tutorial at this address: http://coding.smashingmagazine.com/2008/08/13/top-10-css-table-designs/

This was combined with the tutorial to create responsive tables: shown here: http://css-tricks.com/responsive-data-tables

B. Overall Model

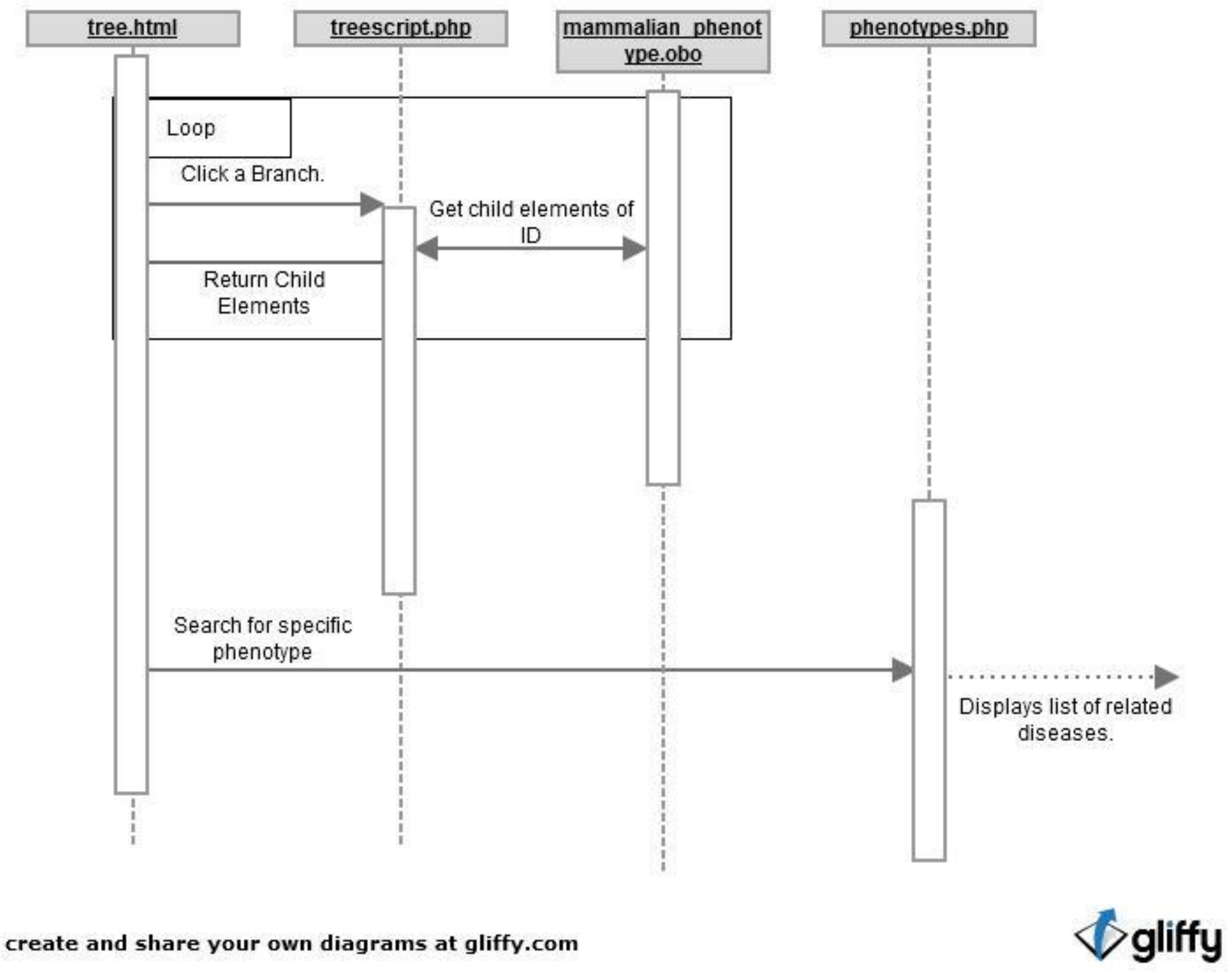


C. Sequence Diagram 1.

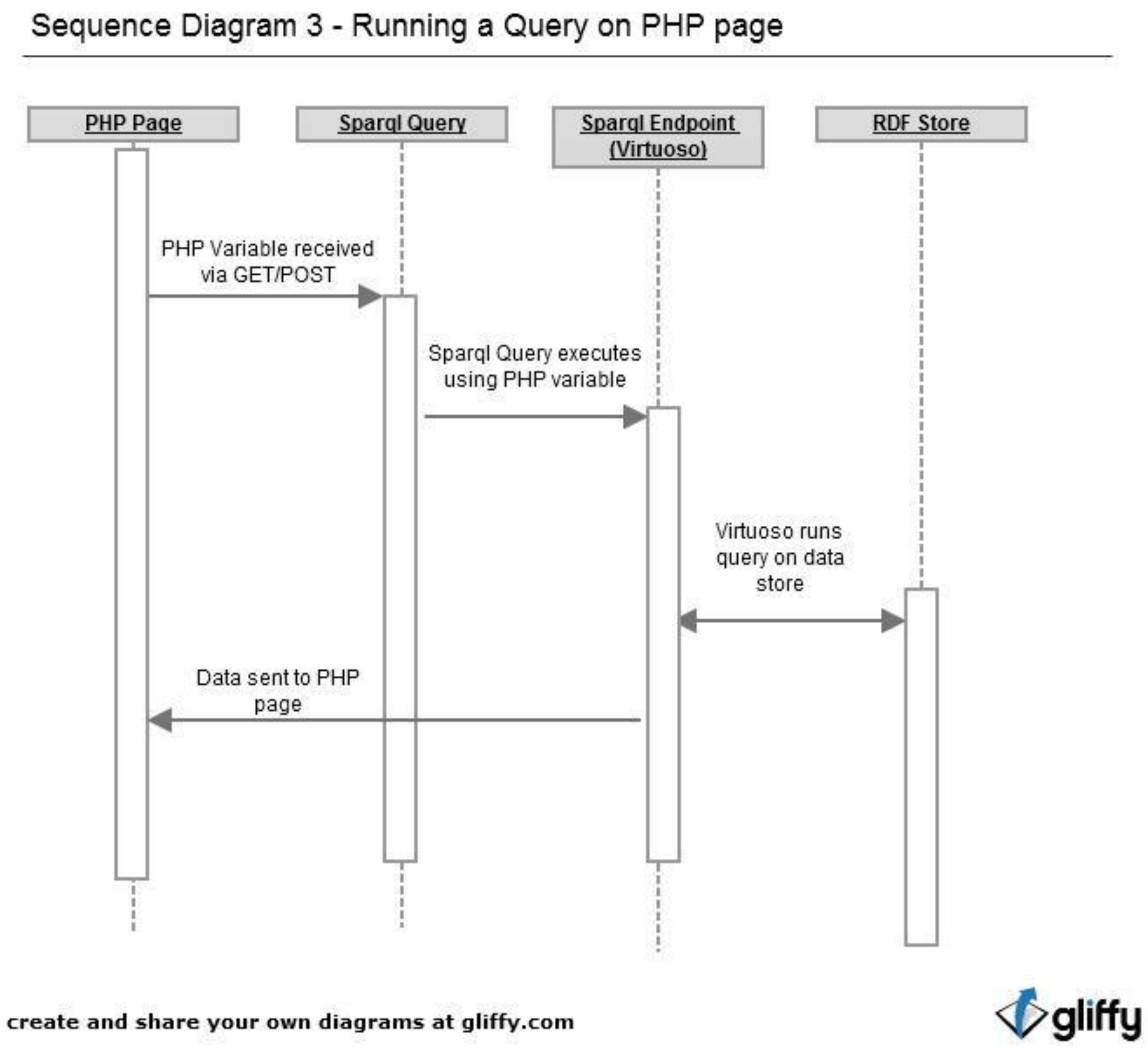

Sequence Diagram 1 - Searching for disease and it's phenotypes
index.html
diseases.php
edge.php
ontologyterms.php
Search for Disease using Form
View related Phenotypes
View related diseases
View related diseases
View related Phenotypes
External site for more information
create and share your own diagrams at gliffy.com
gliffy

D. Sequence Diagram 2.

Sequence Diagram 2 - Search Tree for phenotype and find related diseases



E. Sequence Diagram 3.


Sequence Diagram 3 - Running a Query on PHP page
PHP Page
Sparql Query
Sparql Endpoint (Virtuoso)
RDF Store
PHP Variable received via GET/POST
Sparql Query executes using PHP variable
Virtuoso runs query on data store
Data sent to PHP page
create and share your own diagrams at gliffy.com
gliffy

F. Acceptance testing table.

Test ID	Test Name	Expected Result	Actual Result	Comments
1.	Search for diseases using keyword	User presented with table of diseases, where the disease name contains the keyword searched.	User presented with table of diseases, where the disease name contains the keyword searched.	
2.	Search for gene using keyword	User presented with table of genes, where the gene name contains the keyword searched.	User presented with table of diseases, where the disease name contains the keyword searched.	
3.	View diseases related to chosen disease	User presented with a table of diseases which are related to the chosen disease based on the similarity value shown.	User presented with selection of tables relating to diseases from different backgrounds.	
4.	View genes related to chosen gene	User presented with a list of diseases which are related to the chosen gene based on the similarity value shown.	User presented with selection of tables, containing related diseases or genes, which are split into categories. E.g. OMIM.	Could be done better. Show only data required by user, not all.
5.	View phenotypes related to chosen disease/gene.	Two tables of directly, and inferred phenotypes are displayed.	User presented with selection of tables, containing related diseases or genes, which are split into categories. E.g. MGI	Could be done better. Show only data required by user, not all.
6.	Tree Structure to navigate phenotypes.	Expand and contract branches of tree to search through phenotype ontology.	Expand and contract branches of tree to search through phenotype ontology.	
7.	Use tree structure to	Table of results, showing the diseases	Table of results, showing the diseases	

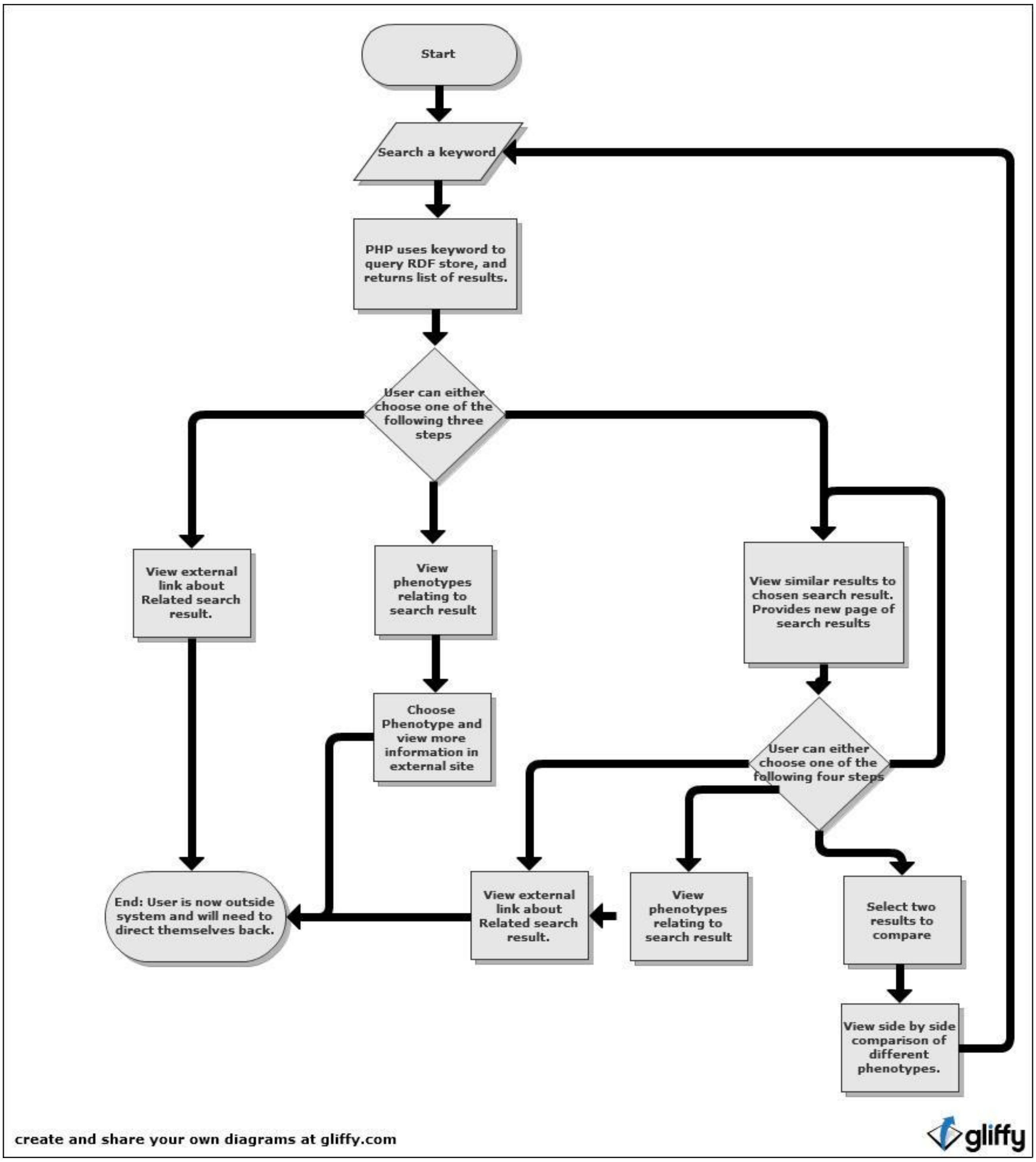
	search for diseases related to specific phenotype	related to the phenotypes	related to the phenotypes	

G. Integration Test Tables

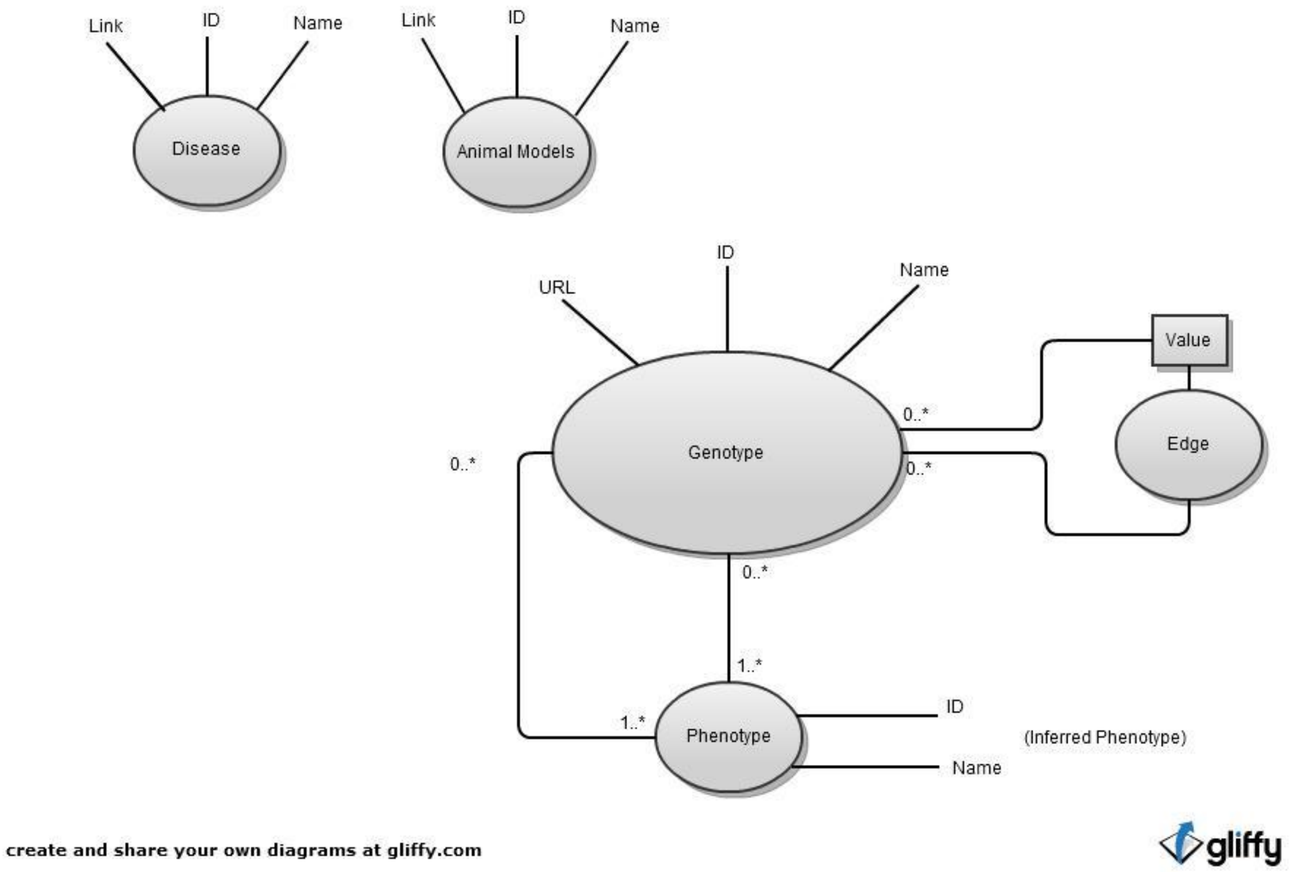
Test ID	Test Name	Expected Result	Actual Result	Comments
1.	Form Validation	Form on Index.html will not submit if it is empty.	Form doesn't submit, and provides an alert box.	Could possibly have more validation to filter malicious attacks.
2.	Variable sent from form to diseases.php	Variable printed to page using echo command.	Variable printed to page using echo command.	
3.	Connect to SPARQL endpoint on virtuoso.	No "failure to connect" error from PHP.	No "failure to connect" error from PHP.	Could be done better. Absence of error message isn't too helpful.
4.	Use PHP variable in SPARQL query.	Table of results relating to variable displayed.	Table of results relating to variable displayed.	
5.	Pass variable from links on diseases.php to edge.php using GET.	URL displays variable in it. Additionally variable printed using echo.	URL displays variable in it. Variable not printed.	Was using POST to retrieve variable. Changed to get and it works.
6.	Pass variable from links on diseases.php to ontologyterms .php using GET.	URL displays variable in it. Additionally variable printed using echo.	URL displays variable in it. Additionally variable printed using echo.	
7.	Load tree structure using parent branches.	Tree loads showing the top level branches. (aka all branches related to MP:0000001 which is the Root.)	Tree loads showing the top level branches	This confirms that all 3 files, tree.html, treescript.php and mammalion_phenotypes.obo are integrated and working in harmony.

8.	Load child branches dependent on which link in tree was clicked.	Children of selected link are loaded, in a tree like format.	Children of selected link are loaded, in a tree like format.	
9.	Pass phenotype ID variable to phenotypes.php	Variable is printed out using echo statement.	Variable is printed out using echo statement.	
10.	Links to external websites from the table of results.	Links to related page dependent on which disease was selected.	Links to error page.	Due to the removal of some of the syntax from the string, it needed to be reinserted to match the website URL. Fixed now to work.
11.	Use hyperlinks on edge.php to move to a position lower down in the page.	Page jumps to appropriate place.	Page jumps to appropriate place.	
12.	Send email using the contact form on contact.html	Email client opens and user can send email.	Email client opens and user can send email.	

H. Initial Flow diagram.


Start
Search a keyword
PHP uses keyword to query RDF store, and returns list of results.
User can either choose one of the following three steps
View external link about Related search result.
View phenotypes relating to search result
View similar results to chosen search result. Provides new page of search results
Choose Phenotype and view more information in external site
User can either choose one of the following four steps
End: User is now outside system and will need to direct themselves back.
View external link about Related search result.
View phenotypes relating to search result
Select two results to compare
View side by side comparison of different phenotypes.
create and share your own diagrams at gliffy.com
gliffy

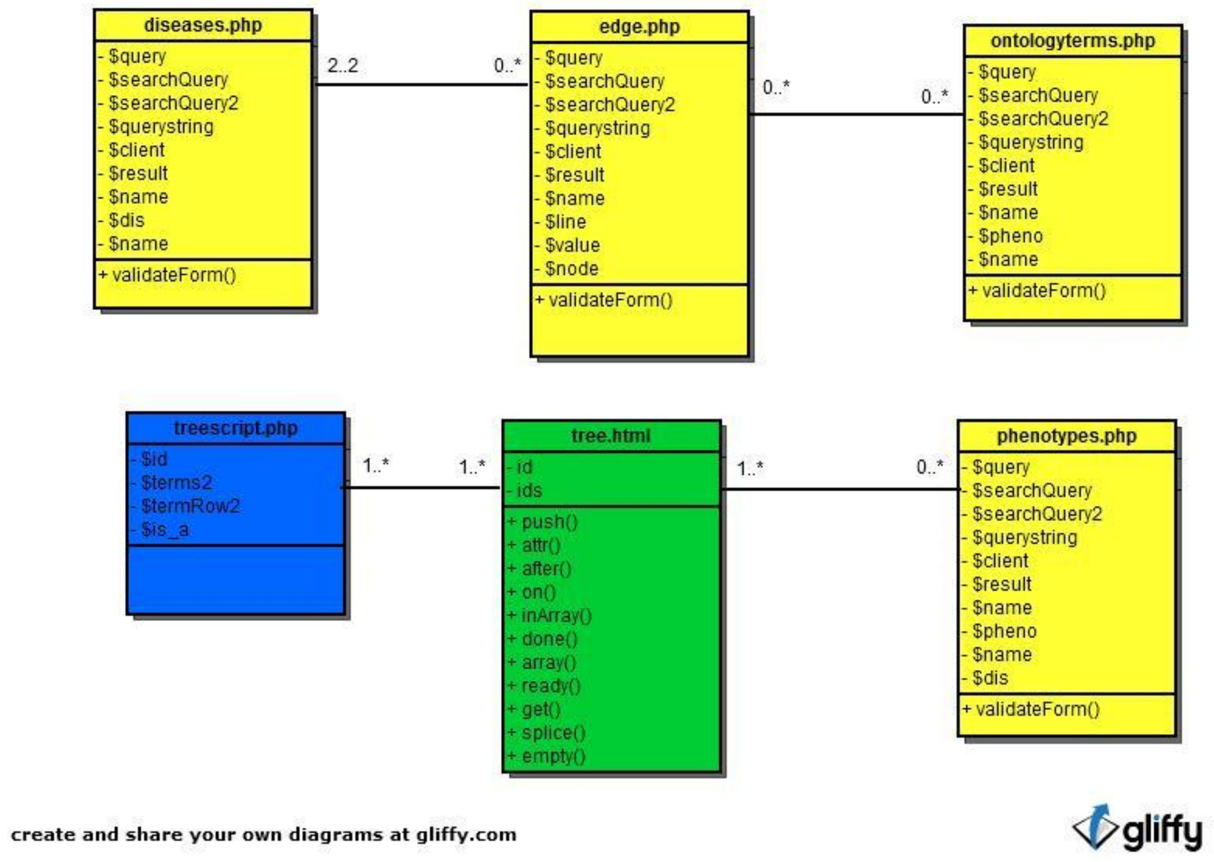
I. Original RDF Schema



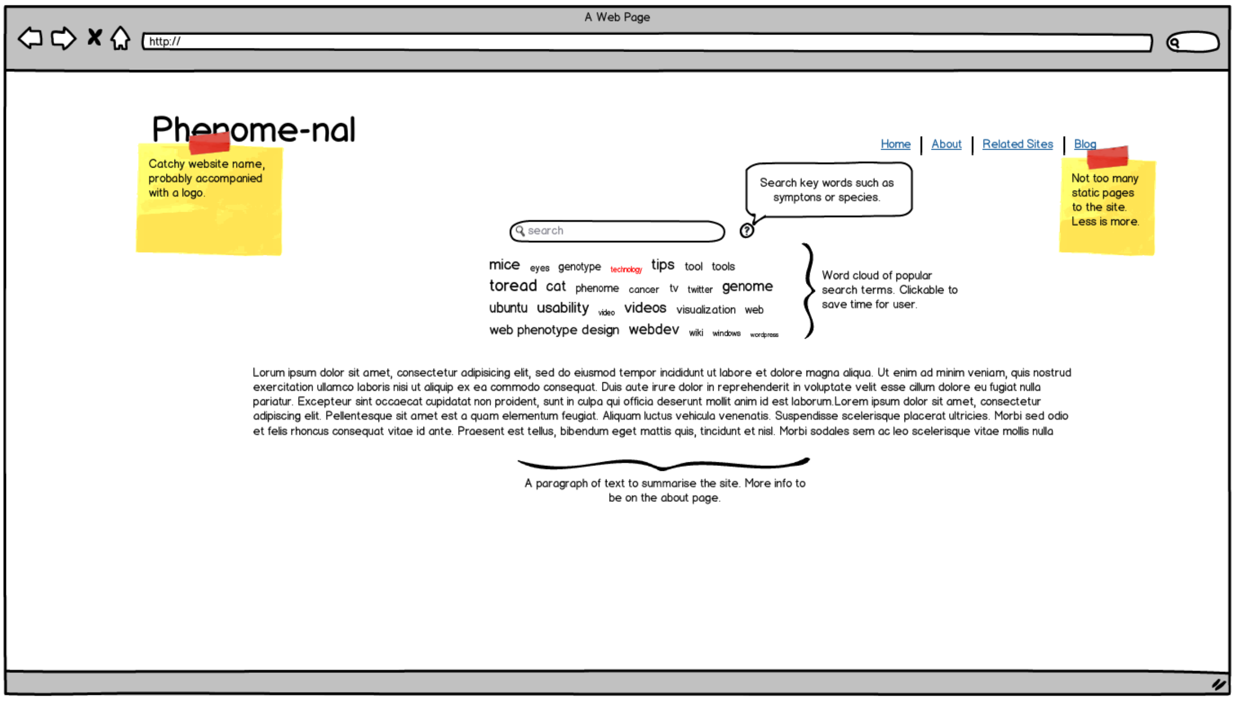
J. Final Schema used during creation of RDF data.

```xml
<?xml version="1.0"?>
<rdf:RDF
xmlns:obo="http://obofoundry.org/obo/"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:phenomenet="http://phenomebrowser.org/phenomenet/">

    <phenomenet:Genotype rdf:about="http://phenomebrowser.org/phenomenet/OMIM_187500">

        <phenomenet:has_url>http://omim.org/entry/187500</phenomenet:has_url>
        <phenomenet:has_name>Tetralogy of Fallot</phenomenet:has_name>

        <phenomenet:has_phenotype rdf:resource="http://obofoundry.org/obo/HP_0000006" />
        <phenomenet:has_phenotype rdf:resource="http://obofoundry.org/obo/HP_0000337" />

        <phenomenet:has_inferred_phenotype rdf:resource="http://obofoundry.org/obo/MP_0000001" />
        <phenomenet:has_inferred_phenotype rdf:resource="http://obofoundry.org/obo/MP_0000266" />

        <phenomenet:has_edge>
            <phenomenet:Edge rdf:about="http://phenomebrowser.org/phenomenet/OMIM_187500_OMIM_187500">
                <phenomenet:has_node rdf:resource="http://phenomebrowser.org/phenomenet/OMIM_187500" />
                <phenomenet:has_node rdf:resource="http://phenomebrowser.org/phenomenet/OMIM_187500" />
                <phenomenet:has_value>1.0</phenomenet:has_value>
            </phenomenet:Edge>
        </phenomenet:has_edge>
        <phenomenet:has_edge rdf:resource="http://phenomebrowser.org/phenomenet/OMIM_187500_OMIM_187501" />
        <phenomenet:has_edge rdf:resource="http://phenomebrowser.org/phenomenet/OMIM_185500_MGI-GT_439" />
    </phenomenet:Genotype>

    <phenomenet:Edge rdf:about="http://phenomebrowser.org/phenomenet/OMIM_187500_MGI-GT_439">
        <phenomenet:has_node rdf:resource="http://phenomebrowser.org/phenomenet/OMIM_187500" />
        <phenomenet:has_node rdf:resource="http://phenomebrowser.org/phenomenet/MGI-GT_439" />
        <phenomenet:has_value>0.389606</phenomenet:has_value>
    </phenomenet:Edge>
    <phenomenet:Ont rdf:about="http://obofoundry.org/obo/MP_0000001">
        <phenomenet:has_name>mammalian phenotype</phenomenet:has_name>
    </phenomenet:Ont>
</rdf:RDF>
```

K. Coloured UML Diagram



L. Original Mock Up Designs


A Web Page
http://
Phenome-nal
Home | About | Related Sites | Blog
search
mice eyes genotype technology tips tool tools toread cat phenome cancer tv twitter genome ubuntu usability video videos visualization web web phenotype design webdev wiki windows wordpress

Catchy website name, probably accompanied with a logo.

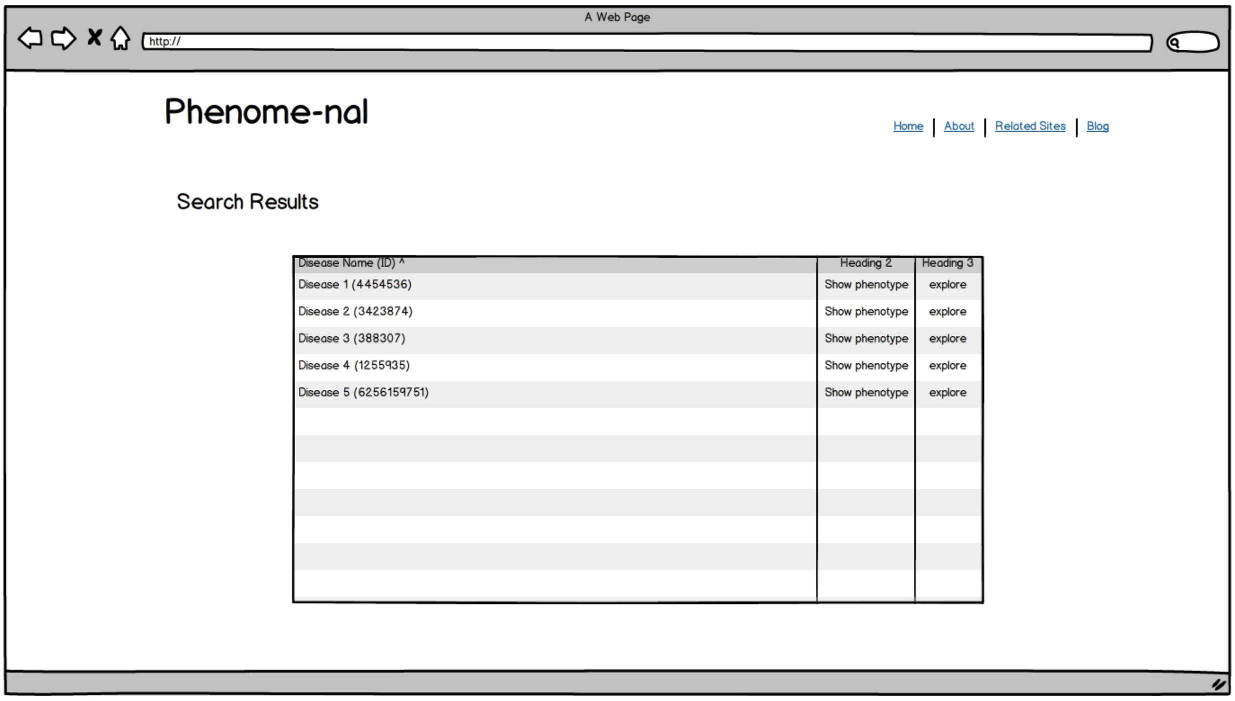
Search key words such as symptons or species.

Not too many static pages to the site. Less is more.

Word cloud of popular search terms. Clickable to save time for user.

Lorum ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque sit amet est a quam elementum feugiat. Aliquam luctus vehicula venenatis. Suspendisse scelerisque placerat ultricies. Morbi sed odio et felis rhoncus consequat vitae id ante. Praesent est tellus, bibendum eget mattis quis, tincidunt et nisl. Morbi sodales sem ac leo scelerisque vitae mollis nulla

A paragraph of text to summarise the site. More info to be on the about page.

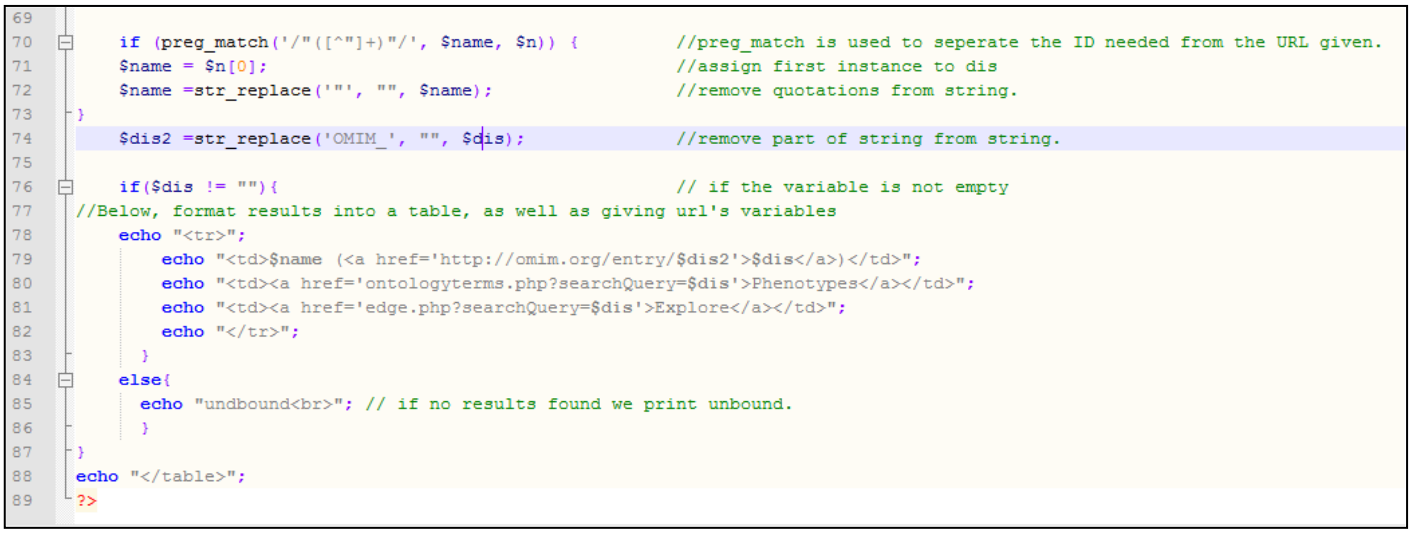


Search Results

Disease Name (ID) ^	Heading 2	Heading 3
Disease 1 (4454536)	Show phenotype	explore
Disease 2 (3423874)	Show phenotype	explore
Disease 3 (388307)	Show phenotype	explore
Disease 4 (1255935)	Show phenotype	explore
Disease 5 (6256159751)	Show phenotype	explore

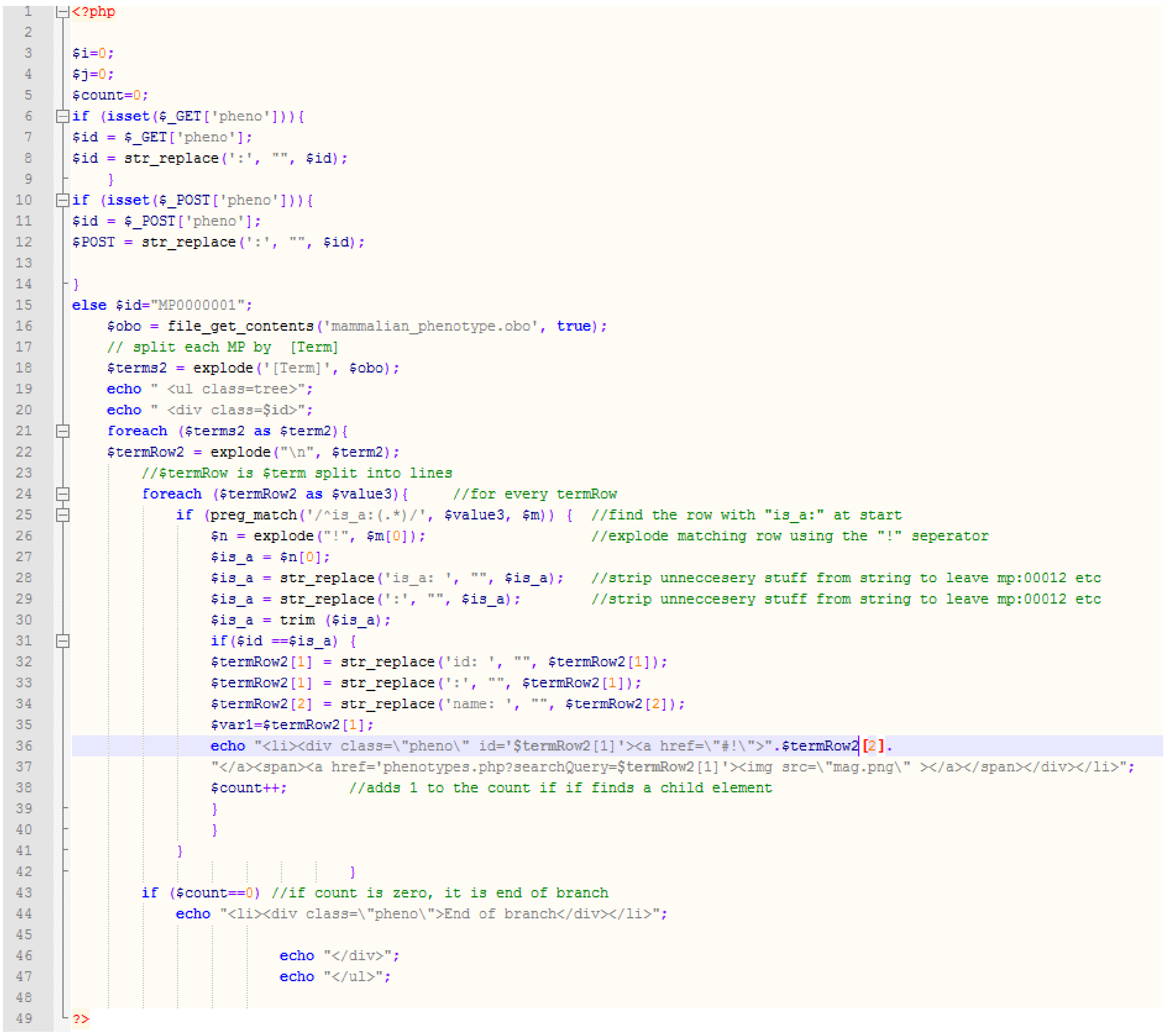
M. Full PHP and SPARQL code. Taken from diseases.php.

```php
<?php
//Gets variable from POST method
    if (isset($_POST['searchQuery'])){
    $searchQuery = $_POST['searchQuery'];

}

//include RAP API RDF library
define("RDFAPI_INCLUDE_DIR", "rdfapi-php/api/");
include(RDFAPI_INCLUDE_DIR . "RdfAPI.php");

//assign endpoint to sparql client
$client = ModelFactory::getSparqlClient("http://biocrunch.dcs.aber.ac.uk:8890/sparql");

//Find the name of all diseases which match the varible passed
$querystring = "
PREFIX phe: <http://phenomebrowser.org/phenomenet/>
PREFIX obo: <http://obofoundry.org/obo>
select *
FROM <http://biocrunch.dcs.aber.ac.uk:8890/DAV/complete>
where {
FILTER regex(?name, '$searchQuery', 'i')
?dis phe:has_name ?name .
FILTER(REGEX(STR(?dis), '^http://phenomebrowser.org/phenomenet'))
}

LIMIT 200";
// ******Comments below refer to query above********
//FROM explains which graph.
//First filter uses the variable to look
//for only diseases with this word in its name. 'i' means case insensitive
//second filter - allows for onyl diseases, and not phenotypes
// Limit - limits to 200 results.



//To execute the query, we create a new ClientQuery
//object and pass it to the SPARQL client:
$query = new ClientQuery();
$query->query($querystring);
$result = $client->query($query);

//The following code loops over the result set and prints out all
//results of the variable
?>
<!-- Table headings -->
 <table id="hor-minimalist-a" summary="Diseases">
 <thead>
        <tr>
            <th scope="col">Disease  name (ID)</th>
            <th scope="col">Phenotype Link</th>
            <th scope="col">Explore</th>
        </tr>
    </thead>
    <tbody>
 <?php
//for every results found.
foreach($result as $line){
// each occurence of ?variable is assigned to a php variable.
  $dis = $line['?dis'];
  $name = $line['?name'];

    if (preg_match('/"([^"]+)"/', $dis, $m)) {          //preg_match is used to seperate the ID needed from the URL given.
    $dis = $m[0];                                       //assign first instance to variable
    $c=explode("/", $dis);                              //explode variable to get just end of url
    $dis=end($c);                                       //gets the end of the array of array
    $dis = str_replace('"', "", $dis);                  //remove quotations from string.
}
```



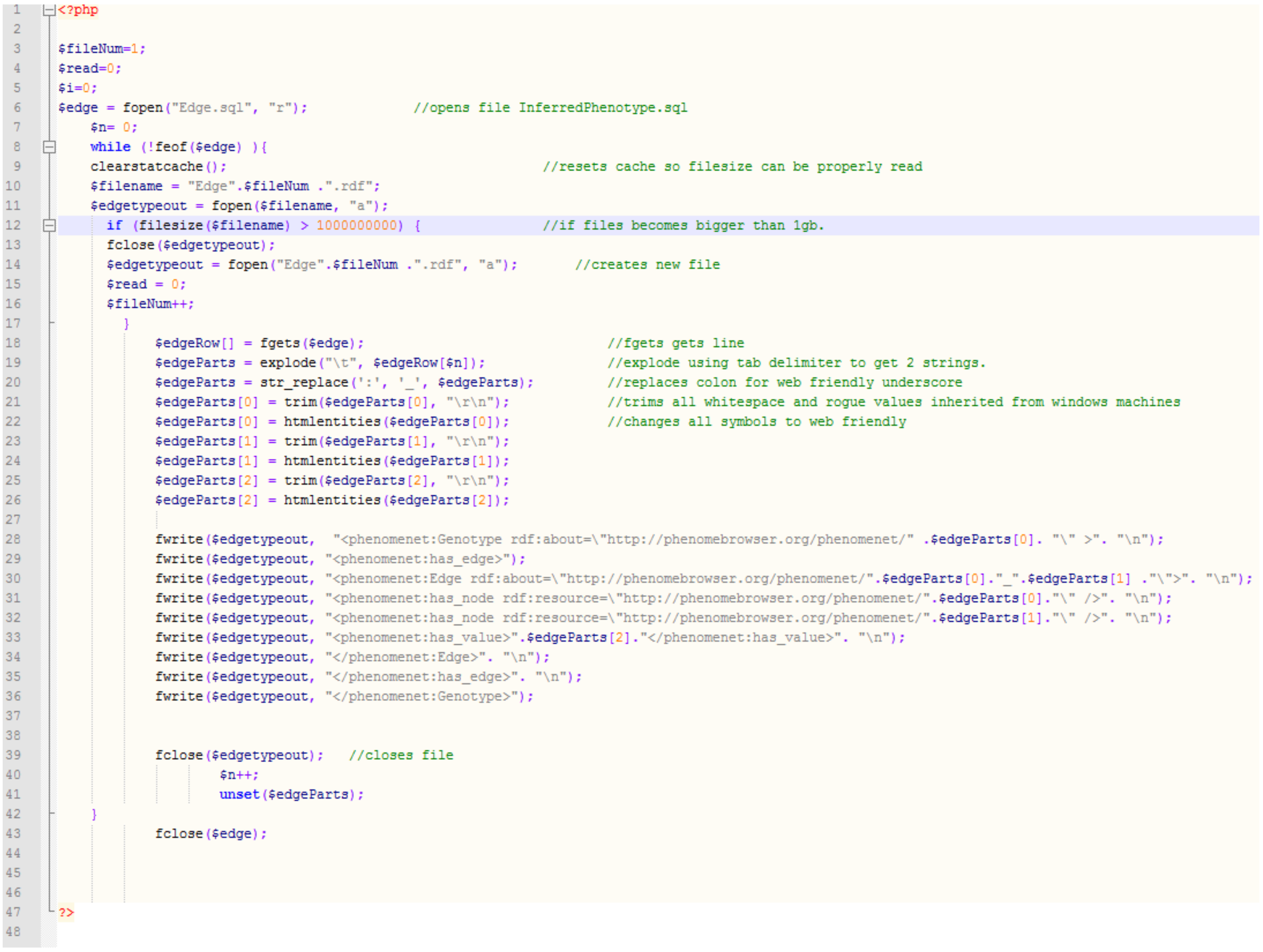
```
    if (preg_match('/"([^"]+)"/', $name, $n)) {        //preg_match is used to seperate the ID needed from the URL given.
    $name = $n[0];                                      //assign first instance to dis
    $name =str_replace('"', "", $name);                 //remove quotations from string.
}
    $dis2 =str_replace('OMIM_', "", $dis);              //remove part of string from string.

    if($dis != ""){                                     // if the variable is not empty
//Below, format results into a table, as well as giving url's variables
    echo "<tr>";
        echo "<td>$name (<a href='http://omim.org/entry/$dis2'>$dis</a>)</td>";
        echo "<td><a href='ontologyterms.php?searchQuery=$dis'>Phenotypes</a></td>";
        echo "<td><a href='edge.php?searchQuery=$dis'>Explore</a></td>";
        echo "</tr>";
      }
    else{
      echo "undbound<br>"; // if no results found we print unbound.
      }
}
echo "</table>";
?>
```

N. Full code and comments from Treescript.php.



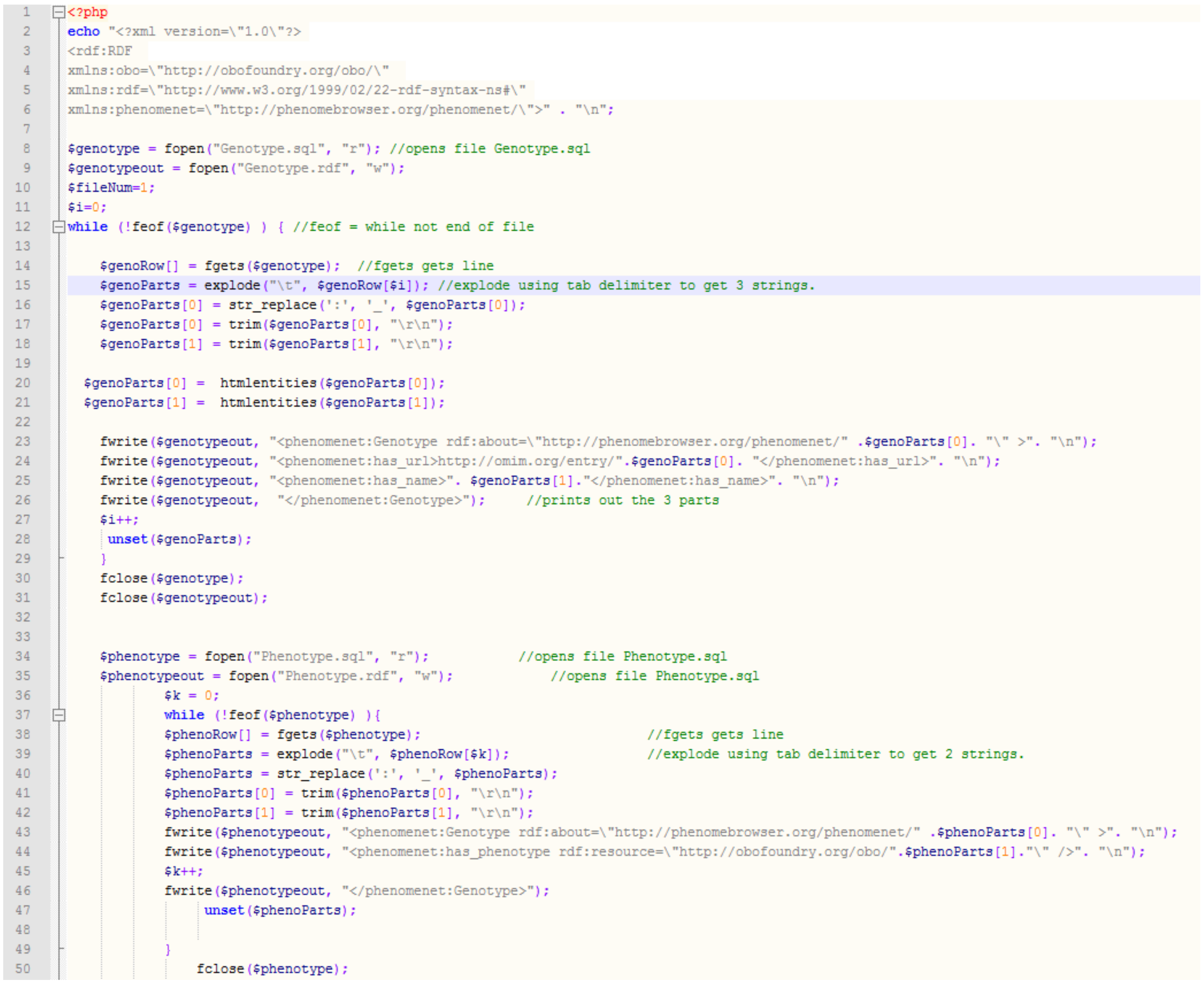
```php
<?php

$i=0;
$j=0;
$count=0;
if (isset($_GET['pheno'])){
$id = $_GET['pheno'];
$id = str_replace(':', "", $id);
    }
if (isset($_POST['pheno'])){
$id = $_POST['pheno'];
$POST = str_replace(':', "", $id);

}
else $id="MP0000001";
    $obo = file_get_contents('mammalian_phenotype.obo', true);
    // split each MP by  [Term]
    $terms2 = explode('[Term]', $obo);
    echo " <ul class=tree>";
    echo " <div class=$id>";
    foreach ($terms2 as $term2){
    $termRow2 = explode("\n", $term2);
        //$termRow is $term split into lines
        foreach ($termRow2 as $value3){     //for every termRow
            if (preg_match('/^is_a:(.*)/', $value3, $m)) {  //find the row with "is_a:" at start
                $n = explode("!", $m[0]);                   //explode matching row using the "!" seperator
                $is_a = $n[0];
                $is_a = str_replace('is_a: ', "", $is_a);   //strip unneccesery stuff from string to leave mp:00012 etc
                $is_a = str_replace(':', "", $is_a);        //strip unneccesery stuff from string to leave mp:00012 etc
                $is_a = trim ($is_a);
                if($id ==$is_a) {
                $termRow2[1] = str_replace('id: ', "", $termRow2[1]);
                $termRow2[1] = str_replace(':', "", $termRow2[1]);
                $termRow2[2] = str_replace('name: ', "", $termRow2[2]);
                $var1=$termRow2[1];
                echo "<li><div class=\"pheno\" id='$termRow2[1]'><a href=\"#!\">".$termRow2[2].
                "</a><span><a href='phenotypes.php?searchQuery=$termRow2[1]'><img src=\"mag.png\" ></a></span></div></li>";
                $count++;       //adds 1 to the count if if finds a child element
                }
                }
            }
                                }
        if ($count==0) //if count is zero, it is end of branch
            echo "<li><div class=\"pheno\">End of branch</div></li>";

                        echo "</div>";
                        echo "</ul>";

?>
```

O. Full Code for Parse Script 1.



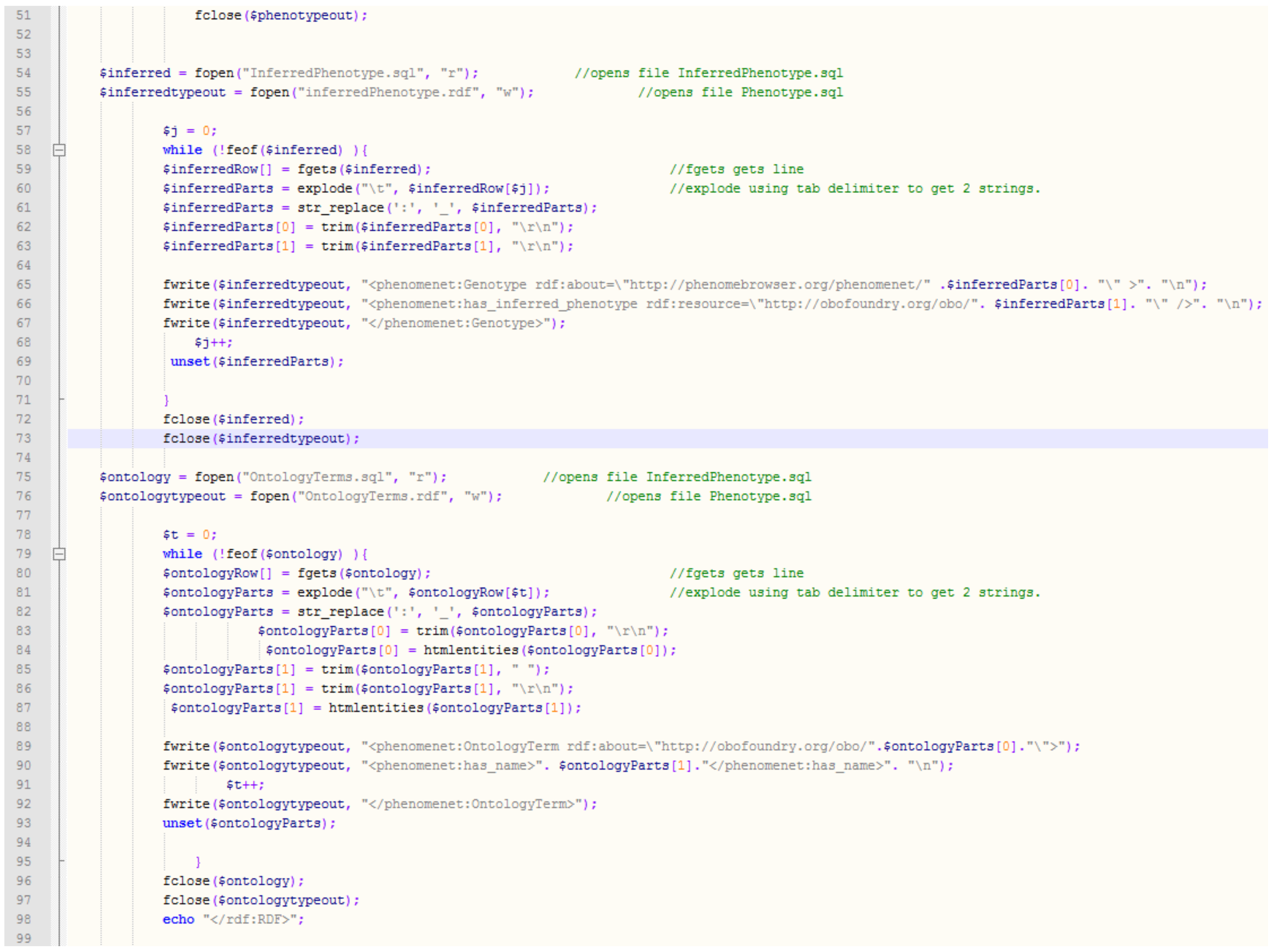
```php
<?php

$fileNum=1;
$read=0;
$i=0;
$edge = fopen("Edge.sql", "r");             //opens file InferredPhenotype.sql
    $n= 0;
    while (!feof($edge) ){
    clearstatcache();                               //resets cache so filesize can be properly read
    $filename = "Edge".$fileNum .".rdf";
    $edgetypeout = fopen($filename, "a");
      if (filesize($filename) > 1000000000) {       //if files becomes bigger than 1gb.
      fclose($edgetypeout);
      $edgetypeout = fopen("Edge".$fileNum .".rdf", "a");       //creates new file
      $read = 0;
      $fileNum++;
        }
            $edgeRow[] = fgets($edge);                          //fgets gets line
            $edgeParts = explode("\t", $edgeRow[$n]);           //explode using tab delimiter to get 2 strings.
            $edgeParts = str_replace(':', '_', $edgeParts);     //replaces colon for web friendly underscore
            $edgeParts[0] = trim($edgeParts[0], "\r\n");        //trims all whitespace and rogue values inherited from windows machines
            $edgeParts[0] = htmlentities($edgeParts[0]);        //changes all symbols to web friendly
            $edgeParts[1] = trim($edgeParts[1], "\r\n");
            $edgeParts[1] = htmlentities($edgeParts[1]);
            $edgeParts[2] = trim($edgeParts[2], "\r\n");
            $edgeParts[2] = htmlentities($edgeParts[2]);

            fwrite($edgetypeout,  "<phenomenet:Genotype rdf:about=\"http://phenomebrowser.org/phenomenet/" .$edgeParts[0]. "\" >". "\n");
            fwrite($edgetypeout, "<phenomenet:has_edge>");
            fwrite($edgetypeout, "<phenomenet:Edge rdf:about=\"http://phenomebrowser.org/phenomenet/".$edgeParts[0]."_".$edgeParts[1] ."\">". "\n");
            fwrite($edgetypeout, "<phenomenet:has_node rdf:resource=\"http://phenomebrowser.org/phenomenet/".$edgeParts[0]."\" />". "\n");
            fwrite($edgetypeout, "<phenomenet:has_node rdf:resource=\"http://phenomebrowser.org/phenomenet/".$edgeParts[1]."\" />". "\n");
            fwrite($edgetypeout, "<phenomenet:has_value>".$edgeParts[2]."</phenomenet:has_value>". "\n");
            fwrite($edgetypeout, "</phenomenet:Edge>". "\n");
            fwrite($edgetypeout, "</phenomenet:has_edge>". "\n");
            fwrite($edgetypeout, "</phenomenet:Genotype>");


            fclose($edgetypeout);   //closes file
                    $n++;
                    unset($edgeParts);
    }
            fclose($edge);



?>
```

P. Full Code for Parse Script 2.



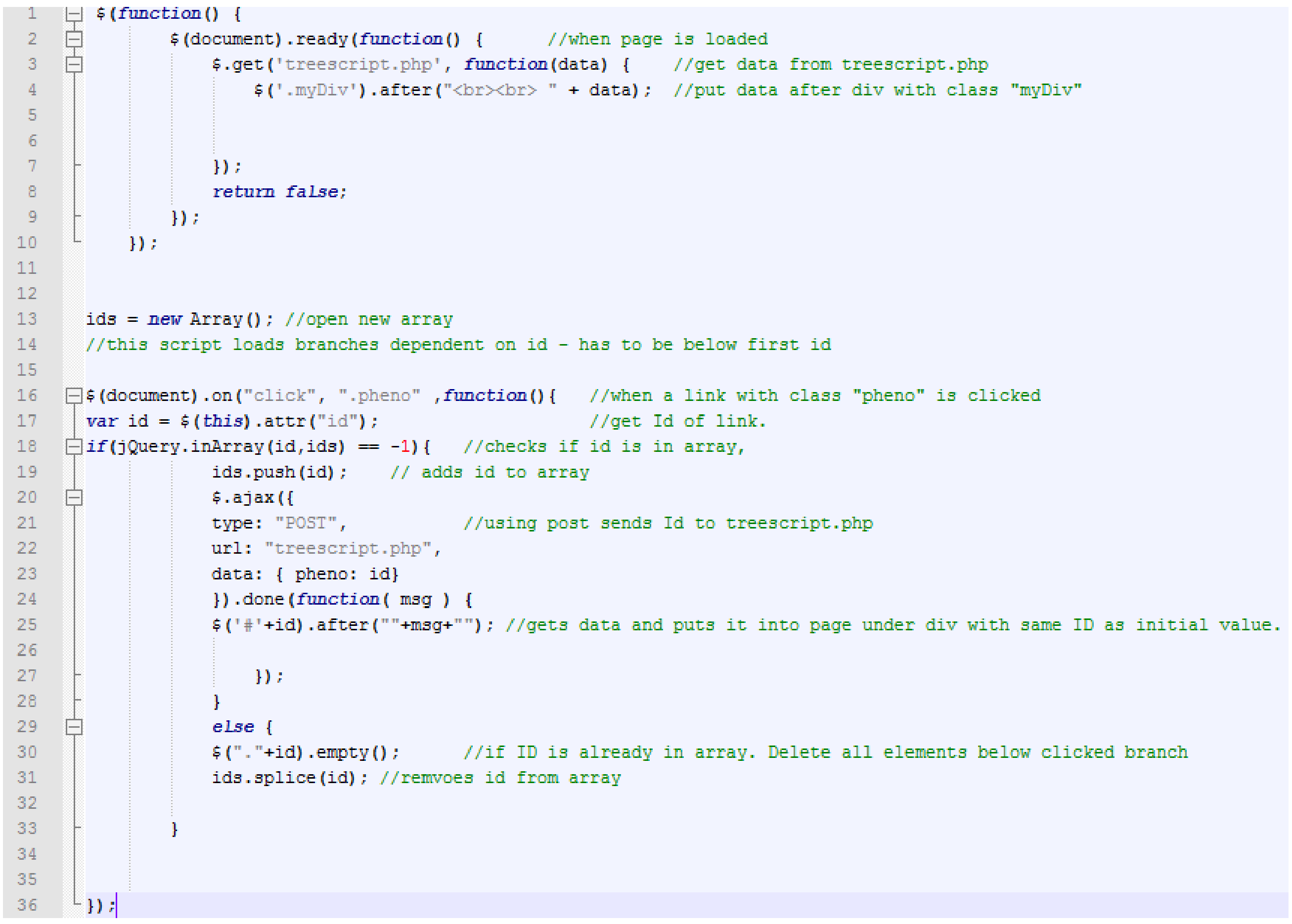
```php
<?php
echo "<?xml version=\"1.0\"?>
<rdf:RDF
xmlns:obo=\"http://obofoundry.org/obo/\"
xmlns:rdf=\"http://www.w3.org/1999/02/22-rdf-syntax-ns#\"
xmlns:phenomenet=\"http://phenomebrowser.org/phenomenet/\">" . "\n";

$genotype = fopen("Genotype.sql", "r"); //opens file Genotype.sql
$genotypeout = fopen("Genotype.rdf", "w");
$fileNum=1;
$i=0;
while (!feof($genotype) ) { //feof = while not end of file

    $genoRow[] = fgets($genotype);  //fgets gets line
    $genoParts = explode("\t", $genoRow[$i]); //explode using tab delimiter to get 3 strings.
    $genoParts[0] = str_replace(':', '_', $genoParts[0]);
    $genoParts[0] = trim($genoParts[0], "\r\n");
    $genoParts[1] = trim($genoParts[1], "\r\n");

  $genoParts[0] =  htmlentities($genoParts[0]);
  $genoParts[1] =  htmlentities($genoParts[1]);

    fwrite($genotypeout, "<phenomenet:Genotype rdf:about=\"http://phenomebrowser.org/phenomenet/" .$genoParts[0]. "\" >". "\n");
    fwrite($genotypeout, "<phenomenet:has_url>http://omim.org/entry/".$genoParts[0]. "</phenomenet:has_url>". "\n");
    fwrite($genotypeout, "<phenomenet:has_name>". $genoParts[1]."</phenomenet:has_name>". "\n");
    fwrite($genotypeout,  "</phenomenet:Genotype>");     //prints out the 3 parts
    $i++;
     unset($genoParts);
    }
    fclose($genotype);
    fclose($genotypeout);


    $phenotype = fopen("Phenotype.sql", "r");           //opens file Phenotype.sql
    $phenotypeout = fopen("Phenotype.rdf", "w");            //opens file Phenotype.sql
            $k = 0;
            while (!feof($phenotype) ){
            $phenoRow[] = fgets($phenotype);                            //fgets gets line
            $phenoParts = explode("\t", $phenoRow[$k]);                 //explode using tab delimiter to get 2 strings.
            $phenoParts = str_replace(':', '_', $phenoParts);
            $phenoParts[0] = trim($phenoParts[0], "\r\n");
            $phenoParts[1] = trim($phenoParts[1], "\r\n");
            fwrite($phenotypeout, "<phenomenet:Genotype rdf:about=\"http://phenomebrowser.org/phenomenet/" .$phenoParts[0]. "\" >". "\n");
            fwrite($phenotypeout, "<phenomenet:has_phenotype rdf:resource=\"http://obofoundry.org/obo/".$phenoParts[1]."\" />". "\n");
            $k++;
            fwrite($phenotypeout, "</phenomenet:Genotype>");
                 unset($phenoParts);

            }
                fclose($phenotype);
```

Q. Full Code for Parse Script 2 (continued).



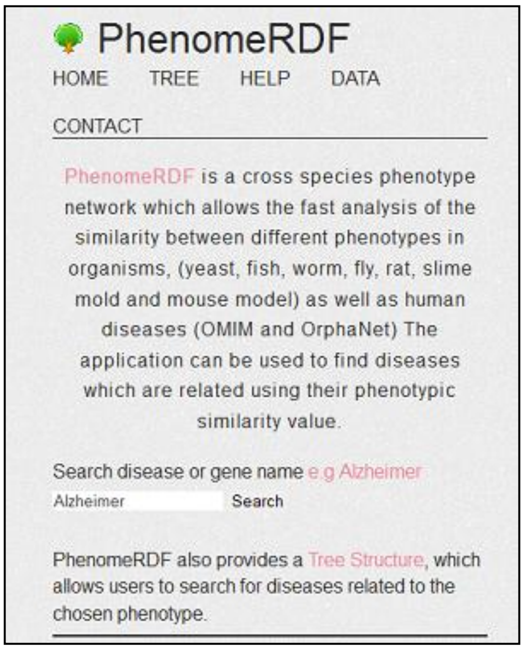
```php
				fclose($phenotypeout);


	$inferred = fopen("InferredPhenotype.sql", "r");			//opens file InferredPhenotype.sql
	$inferredtypeout = fopen("inferredPhenotype.rdf", "w");			//opens file Phenotype.sql

			$j = 0;
			while (!feof($inferred) ){
			$inferredRow[] = fgets($inferred);					//fgets gets line
			$inferredParts = explode("\t", $inferredRow[$j]);			//explode using tab delimiter to get 2 strings.
			$inferredParts = str_replace(':', '_', $inferredParts);
			$inferredParts[0] = trim($inferredParts[0], "\r\n");
			$inferredParts[1] = trim($inferredParts[1], "\r\n");

			fwrite($inferredtypeout, "<phenomenet:Genotype rdf:about=\"http://phenomebrowser.org/phenomenet/" .$inferredParts[0]. "\" >". "\n");
			fwrite($inferredtypeout, "<phenomenet:has_inferred_phenotype rdf:resource=\"http://obofoundry.org/obo/". $inferredParts[1]. "\" />". "\n");
			fwrite($inferredtypeout, "</phenomenet:Genotype>");
				$j++;
			 unset($inferredParts);

			}
			fclose($inferred);
			fclose($inferredtypeout);

	$ontology = fopen("OntologyTerms.sql", "r");			//opens file InferredPhenotype.sql
	$ontologytypeout = fopen("OntologyTerms.rdf", "w");			//opens file Phenotype.sql

			$t = 0;
			while (!feof($ontology) ){
			$ontologyRow[] = fgets($ontology);					//fgets gets line
			$ontologyParts = explode("\t", $ontologyRow[$t]);			//explode using tab delimiter to get 2 strings.
			$ontologyParts = str_replace(':', '_', $ontologyParts);
						$ontologyParts[0] = trim($ontologyParts[0], "\r\n");
						 $ontologyParts[0] = htmlentities($ontologyParts[0]);
			$ontologyParts[1] = trim($ontologyParts[1], " ");
			$ontologyParts[1] = trim($ontologyParts[1], "\r\n");
			 $ontologyParts[1] = htmlentities($ontologyParts[1]);

			fwrite($ontologytypeout, "<phenomenet:OntologyTerm rdf:about=\"http://obofoundry.org/obo/".$ontologyParts[0]."\">");
			fwrite($ontologytypeout, "<phenomenet:has_name>". $ontologyParts[1]."</phenomenet:has_name>". "\n");
					$t++;
			fwrite($ontologytypeout, "</phenomenet:OntologyTerm>");
			unset($ontologyParts);

				}
			fclose($ontology);
			fclose($ontologytypeout);
			echo "</rdf:RDF>";
```

R. jQuery algorithms for Tree.html.



```
 $(function() {
        $(document).ready(function() {      //when page is loaded
            $.get('treescript.php', function(data) {    //get data from treescript.php
                $('.myDiv').after("<br><br> " + data);  //put data after div with class "myDiv"


            });
            return false;
        });
    });


ids = new Array(); //open new array
//this script loads branches dependent on id - has to be below first id

$(document).on("click", ".pheno" ,function(){   //when a link with class "pheno" is clicked
var id = $(this).attr("id");                    //get Id of link.
if(jQuery.inArray(id,ids) == -1){   //checks if id is in array,
            ids.push(id);    // adds id to array
            $.ajax({
            type: "POST",           //using post sends Id to treescript.php
            url: "treescript.php",
            data: { pheno: id}
            }).done(function( msg ) {
            $('#'+id).after(""+msg+""); //gets data and puts it into page under div with same ID as initial value.

                });
            }
            else {
            $("."+id).empty();      //if ID is already in array. Delete all elements below clicked branch
            ids.splice(id); //remvoes id from array

        }


});
```

S. Process of searching for a disease, and viewing the related phenotypes.



PhenomeRDF

HOME TREE HELP DATA

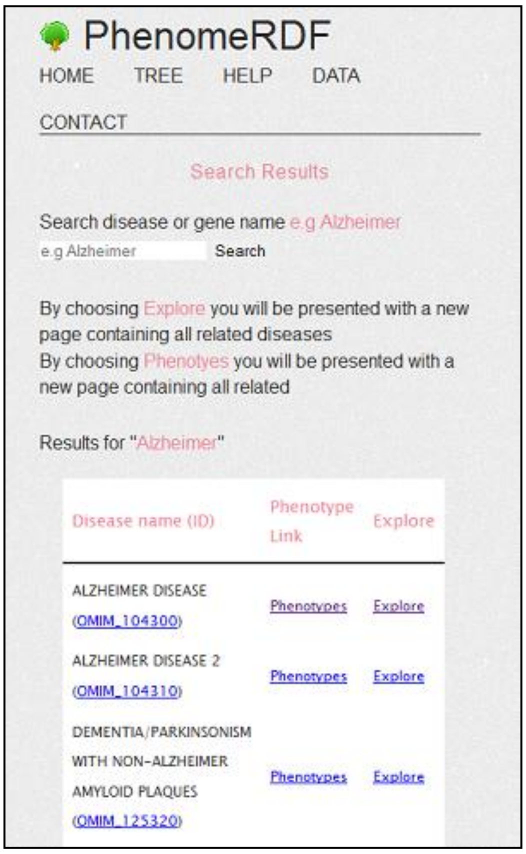
CONTACT

PhenomeRDF is a cross species phenotype network which allows the fast analysis of the similarity between different phenotypes in organisms, (yeast, fish, worm, fly, rat, slime mold and mouse model) as well as human diseases (OMIM and OrphaNet) The application can be used to find diseases which are related using their phenotypic similarity value.

Search disease or gene name e.g Alzheimer

Alzheimer Search

PhenomeRDF also provides a Tree Structure, which allows users to search for diseases related to the chosen phenotype.



PhenomeRDF

HOME TREE HELP DATA

CONTACT

Search Results

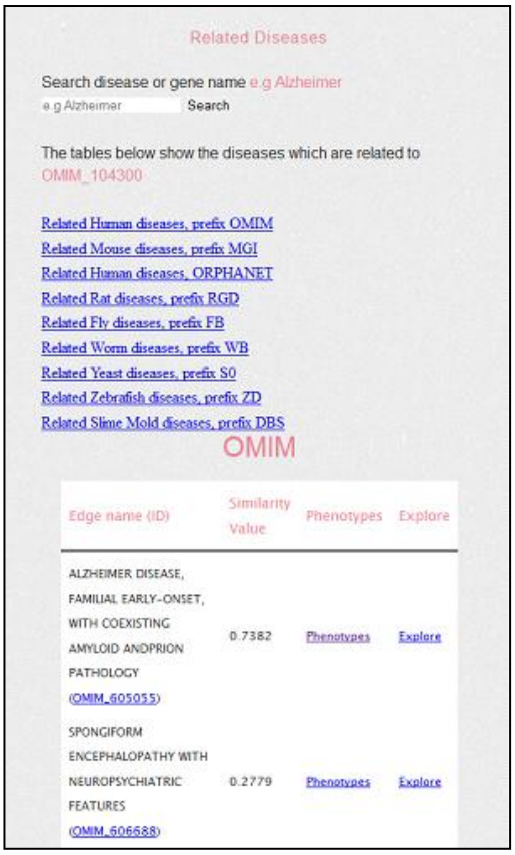
Search disease or gene name e.g Alzheimer

e.g Alzheimer Search

By choosing Explore you will be presented with a new page containing all related diseases
By choosing Phenotyes you will be presented with a new page containing all related

Results for "Alzheimer"

Disease name (ID)	Phenotype Link	Explore
ALZHEIMER DISEASE (OMIM_104300)	Phenotypes	Explore
ALZHEIMER DISEASE 2 (OMIM_104310)	Phenotypes	Explore
DEMENTIA/PARKINSONISM WITH NON-ALZHEIMER AMYLOID PLAQUES (OMIM_125320)	Phenotypes	Explore



Related Diseases

Search disease or gene name e.g Alzheimer

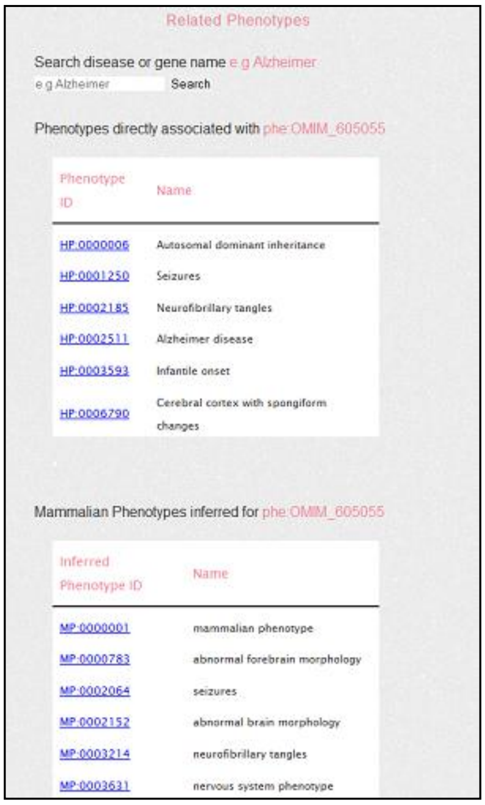
e.g Alzheimer Search

The tables below show the diseases which are related to OMIM_104300

Related Human diseases, prefix OMIM
Related Mouse diseases, prefix MGI
Related Human diseases, ORPHANET
Related Rat diseases, prefix RGD
Related Fly diseases, prefix FB
Related Worm diseases, prefix WB
Related Yeast diseases, prefix S0
Related Zebrafish diseases, prefix ZD
Related Slime Mold diseases, prefix DBS

OMIM

Edge name (ID)	Similarity Value	Phenotypes	Explore
ALZHEIMER DISEASE, FAMILIAL EARLY-ONSET, WITH COEXISTING AMYLOID ANDPRION PATHOLOGY (OMIM_605055)	0.7382	Phenotypes	Explore
SPONGIFORM ENCEPHALOPATHY WITH NEUROPSYCHIATRIC FEATURES (OMIM_606688)	0.2779	Phenotypes	Explore



Related Phenotypes

Search disease or gene name e.g Alzheimer

e.g Alzheimer Search

Phenotypes directly associated with phe:OMIM_605055

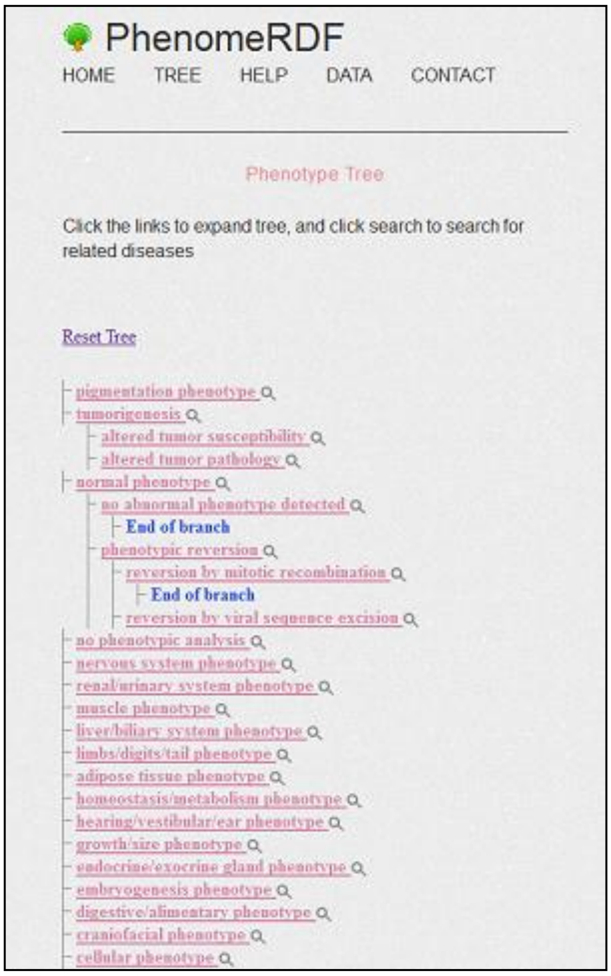
Phenotype ID	Name
HP:0000006	Autosomal dominant inheritance
HP:0001250	Seizures
HP:0002185	Neurofibrillary tangles
HP:0002511	Alzheimer disease
HP:0003593	Infantile onset
HP:0006790	Cerebral cortex with spongiform changes

Mammalian Phenotypes inferred for phe:OMIM_605055

Inferred Phenotype ID	Name
MP:0000001	mammalian phenotype
MP:0000783	abnormal forebrain morphology
MP:0002064	seizures
MP:0002152	abnormal brain morphology
MP:0003214	neurofibrillary tangles
MP:0003631	nervous system phenotype

Figure 11. From top left, clockwise.
1) A user searches for the keyword Alzheimer.
2)Clicks Explore on first result.
3) Clicks phenotypes on first result.
4)Table of related phenotypes.

T. Using the tree structure to search for diseases relating to specific phenotype.

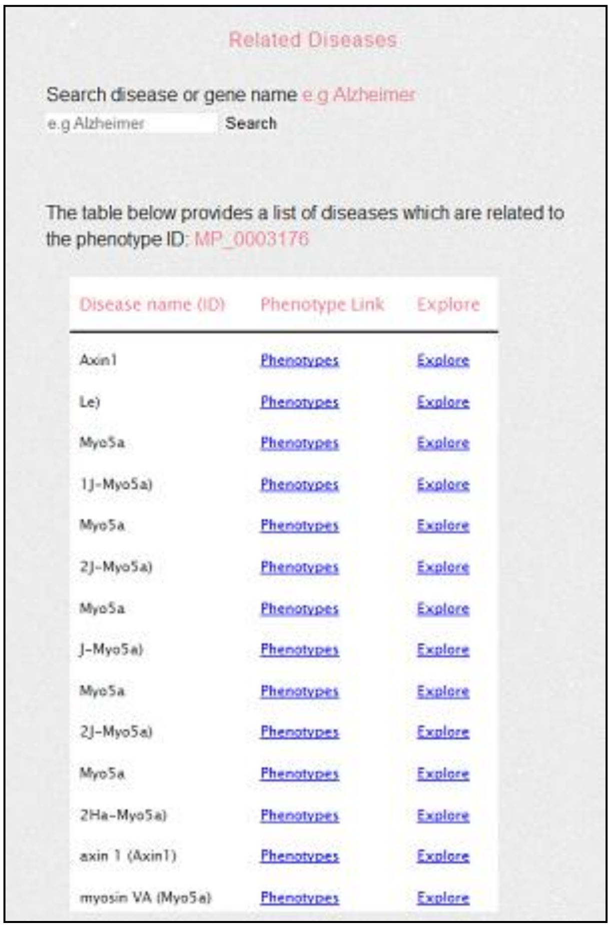


Phenotype Tree

Click the links to expand tree, and click search to search for related diseases

Reset Tree

- pigmentation phenotype
- tumorigenesis
  - altered tumor susceptibility
  - altered tumor pathology
- normal phenotype
  - no abnormal phenotype detected
    - End of branch
  - phenotypic reversion
    - reversion by mitotic recombination
      - End of branch
    - reversion by viral sequence excision
- no phenotypic analysis
- nervous system phenotype
- renal/urinary system phenotype
- muscle phenotype
- liver/biliary system phenotype
- limbs/digits/tail phenotype
- adipose tissue phenotype
- homeostasis/metabolism phenotype
- hearing/vestibular/ear phenotype
- growth/size phenotype
- endocrine/exocrine gland phenotype
- embryogenesis phenotype
- digestive/alimentary phenotype
- craniofacial phenotype
- cellular phenotype


Related Diseases

Search disease or gene name e.g Alzheimer

The table below provides a list of diseases which are related to the phenotype ID: MP_0003176

Disease name (ID)	Phenotype Link	Explore
Axin1	Phenotypes	Explore
Le)	Phenotypes	Explore
Myo5a	Phenotypes	Explore
1J-Myo5a)	Phenotypes	Explore
Myo5a	Phenotypes	Explore
2J-Myo5a)	Phenotypes	Explore
Myo5a	Phenotypes	Explore
J-Myo5a)	Phenotypes	Explore
Myo5a	Phenotypes	Explore
2J-Myo5a)	Phenotypes	Explore
Myo5a	Phenotypes	Explore
2Ha-Myo5a)	Phenotypes	Explore
axin 1 (Axin1)	Phenotypes	Explore
myosin VA (Myo5a)	Phenotypes	Explore

Figure 12. Left to right. A user expands branches in the tree, to find a phenotype of their choice. After clicking the search icon, they are presented with a list of diseases related to the phenotype they chose.

Annotated Bibliography

Chris Biron. A website which provides free CSS layouts. (http://www.free-css.com/assets/files/free-css-templates/preview/page161/liquid-gem/). Copyright 2012. Accessed February 2013.

Chris Bizer and Andreas Schultz, A website which shows the performance of various data stores in different benchmark tests. (http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/results/V6/index.html) February 2011. Accessed November 2012.

Dan Zambonini, A blog which talks about the differences between XML and RDF. (http://www.oreillynet.com/xml/blog/2005/09/the_difference_between_xml_and.html) September 20, 2005 10:41AM. Accessed January 2013.

Daniel Westphal, Chris Bizer, A website which has tutorials on using SPARQL with PHP, as well as other information about the library. (http://wifo5-03.informatik.uni-mannheim.de/bizer/rdfapi/)October 2004. Accessed November 2012.

Joshua Tauberer. A website which describes what RDF is, and how to use it. (http://www.rdfabout.com/, 2005. Accessed November 2012.)

jQuery. This website contains all documentation on how to use the jQuery library as well as every method. (http://api.jquery.com/), April 2013. Accessed March 2013.

Neil Taylor. Powerpoint Presentation detailing the Feature Driven Development lifecycle. Includes a section describing the FDD Coloured UML. 2012. Accessed 2013.

Robert Hoehndorf, Paul N. Schofield and Georgios V. Gkoutos. PhenomeNET: a whole-phenome approach to disease gene discovery. (http://phenomebrowser.net/) Nucleid Acids Research, 2011. Accessed October 2012

Robert Hoehndorf, Paul N. Schofield and Georgios V. Gkoutos. Phenotype ontologies for mouse and man: bridging the semantic gap. (http://dmm.biologists.org/content/3/5-6/281.abstract?ijkey=df3badfe3a581762a9e459ef05fccb8e7b915f73&keytype2=tf_ipsecsha) 2010. Accessed April 2013.

Sheo Narayan . A website describing what jQuery is, and how to use it. (http://www.codeproject.com/Articles/157446/What-is-jQuery-and-How-to-Start-using-jQuery), February 2011. Accessed March 2013.

Tom Heath. A website which discusses what Linked Data is. (http://linkeddata.org/). No date given. Accessed March 2013.

Tim Berners Lee. A website which talks about the difference between RDF and XML. (http://www.w3.org/DesignIssues/RDF-XML.html). September 1998. Accessed March 2013.

PHP Manual. This website contains information on how to install PHP. As well as vast documentation on every method in the PHP library. (http://www.php.net/manual/en/index.php) April 2014. Accessed, April 2014.

Virtuoso. The home page for Virtuoso. (http://virtuoso.openlinksw.com/) Copyright, OpenLink Software 2013. Accessed November 2012.