BIO727P – BIOINFORMATICS SOFTWARE DEVELOPMENT

GROUP PROJECT

Team Falsone Documentation

Feb 2019

Josephine Mensah-Kane 180928329

Ishack Mougamadou 150346599

Richard Burns 180601400

Fatma Altamimi 180822555

Table of Contents

Project Background:

Team Falsone is a 4-member team of MSc in Bioinformatics students at the School of Biological Sciences and Chemistry at QMUL. For this project, our team presents an online web interface for database searching of kinases, the phosphosites they relate to and kinase inhibitors. More importantly, it allows users to upload experimental data for phosphoproteomics analysis. The following document provides a detailed description of how the software was developed and what features it includes.

Software General Description:

Team Falsone aims to create an easy-to-use data resource website containing information on all known human kinases, inhibitors of these kinases and providing the ability for a user to upload their own results to process and present a summary of the activity of human kinases acted on by inhibitors.

The website supplies the user with information on all human protein kinases (name, gene symbol, subcellular location, and all sites they phosphorylate) as well as information on kinase inhibitors (name, chemical structure, chemical formula, and the kinases they are known to inhibit).

All scripts, codes, templates, and relevant information is shared on our GitHub repository (Falsone_Group_Project). In addition, we have successfully launched our website on the Amazon Elastic Beanstalk at the following URL: http://falsone-kinase-browser.eu-west-2.elasticbeanstalk.com.

In figure 1 below, the main web page layout is depicted.

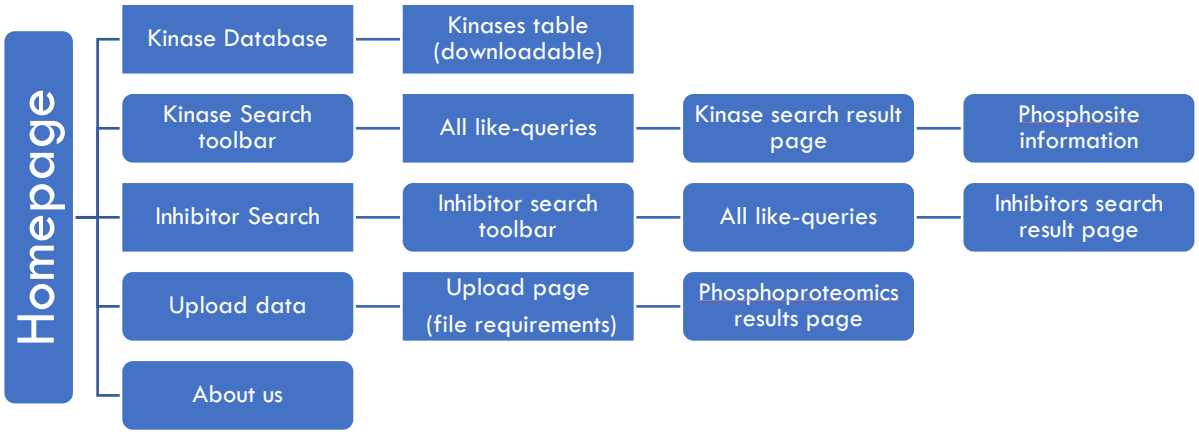

Figure 1. Falsone website navigation map.

On the homepage, a search toolbar allows users to search for any human kinase by protein name, accession number, or gene name. If the user types in a partial query, a 'similar queries' page shows all human kinases from our database in which that partial query string appears, with hyperlinks. From this page, a user can click on those like-matches for the specific query intended and go to the query result page which shows all relevant information of the human kinase as well as information on the sites it is known to phosphorylate, and inhibitors if any. Thus, a user is able to find information on the phosphosites in the same results page as the kinase it relates to.

Inhibitor information may be shown on the kinase search page for that specific kinase, where available. However, a separate inhibitor search feature can be accessed from the homepage in which a user inputs the inhibitor name and gets the relevant results page.

Also, from the homepage, quantitative phosphoproteomics data can be uploaded, and the user is then redirected to the results page of the analysis which

displays descriptive graphics. Furthermore, a short introduction on the team and website features are available on the homepage.

Information Resourcing

The information required for setting up the website was researched and acquired from several resources using different approaches, most of which involved Python coding.

- Human Kinases

Data on human kinases was compiled from the "Human and mouse protein kinases: classification and index" documentation available on the Universal Protein Resource (Uniprot.org, 2019). This document contains all known human and mouse kinases. The data from this document was filtered using a python code to find human kinases only. Once the desired list of all human kinases was gathered, another python coding approach was used to query these kinases into the uniport database through the application programming interface (API) in UniProt. This approach was taken to find the protein kinase name, accession number, gene symbol, gene family name, subcellular location and alternative names.

- Phosphosites

To gather information on kinase-substrate relationships, the 'kinase-substrate dataset' and the 'phosphosphorylation sites dataset' were imported from PhosphoSitePlus (Phosphosite.org, 2019). These data contained all known kinases from human and non-human organisms. Thus, it was necessary that the raw data was filtered to keep the human kinases only. Once the datasets were

filtered, they were both incorporated into our Falsone database. More details about these relational tables will be discussed later in the Building database section.

- Kinase Inhibitors

Information on kinase inhibitors was obtained from the International Centre for Kinase profiling within the MRC Protein Phosphorylation Unit at the University of Dundee (Kinase-screen.mrc.ac.uk, 2019). From this data source, two datasets were obtained. One file includes the bulk information on the inhibitor's identity. The other dataset includes information on the inhibitor and the substrates upon which it acts. From the first data set, the inhibitor name, chemical formula, PubChem ID, and Cnumber were retained. The Cnumber was crucial to linking the kinase inhibitors to the sites they act on as they are only referred to by Cnumber in the second dataset. The PubChem ID was used to retrieve the chemical structure images from another source, PubChem (Pubchem.ncbi.nlm.nih.gov, 2019). Using python coding and the PubChem ID, we were able to achieve URL-based retrieval of an image of the chemical structure for those that have one documented on the PubChem website. These were then incorporated into our database to display the chemical structures on the website.

Building the Falsone Database:

Once all the required relevant information was acquired, a plan was laid out to account for the links between different datasets in order to build the Falsone Database. The relational database was therefore laid out to best store information on kinases and their relationships with inhibitors, substrates and modified residue phosphosites, as well as the relationships between kinases and inhibitors. Written in SQLite3, this relational database was curated in DB Browser for SQLite, where the schema was loaded and the data imported from csv files. Figure 2 below shows all the tables created in the Falsone Database with the primary keys in each table and the relational arrows.

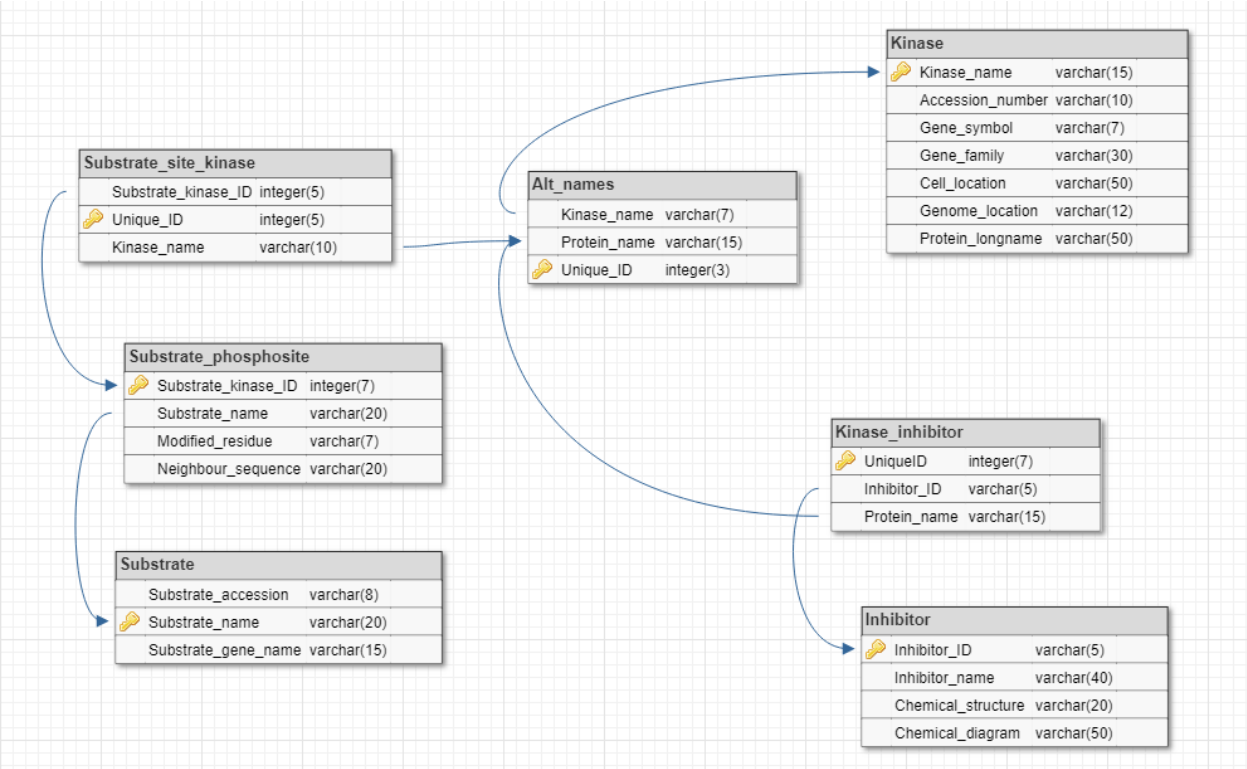

Figure 2 Falsone Database schematic diagram of tables and their relationships

The first main table in the database; 'Kinase', stores the main bulk of information on all known human kinases. This table includes name, UniProt

accession number, gene symbol, gene family, cellular location, genomic location and protein long name. The primary key of this table is the kinase name, which is unique per entry, and this table has no foreign key dependencies.

The second table ‘Alt names’ was a direct offshoot of the ‘Kinase’ table. There was a recurrent issue within the data in which a kinase was often not referred to by its agreed kinase name and was instead either referred to by any of its many alternative names, long names or gene symbols. This information all needed to be stored in the database, so that inhibitor or substrate data that didn’t directly refer to a kinase name could be linked back to its corresponding kinase in the ‘Kinase’ table.

The ‘Alt names' table was simply laid out with fields for a kinase name, a corresponding alternative name and a unique ID (which formed the primary key). Any kinase could be present in the table as many times as there were alternative names for it, and there were also some minor redundancies in which an alternative name could be mapped back to two different kinases (which is a scientific integrity issue, data should not be referred to by names which do not directly map back to one specific entity). In this table a foreign key existed for the kinase name field, referencing the kinase names field in ‘Kinase', ensuring that every kinase in the alternative names list was actually referring to a human kinase.

Each kinase in the ‘Kinase’ table had multiple sites that it is known to phosphorylate on different substrates. All of this data also needed to be stored within the database. In order to fully normalise the schema and ensure minimal

redundancies in the database, substrate information is distributed between three different tables; one with information on each substrate, one for information on each phosphosite present for each substrate, and one relationship table mapping the substrate phosphosites to at least one kinase.

The ‘Substrate’ table is also laid out, with fields for the UniProt accession number of each substrate in the data, along with the name of the substrate and its main gene name. With no foreign key dependencies, the primary key in this table is the substrate name.

The ‘Substrate phosphosite' table is designed to give information for each phosphosite available for the substrates in the substrate table. This information includes the substrate name, modified residue in the substrate, the position at which it is modified, and the neighbouring sequence surrounding the modified residue. This table also assigns a substrate-kinase-ID which links a substrate name/site combination to a kinase via a relationship table. In this table, the substrate name is stored again. However, it was sub-optimal to have this, and the ‘substrate' table merged into one, as the same names, symbols and accessions were all repeated multiple times throughout the table, while in this schema it is just substrate name which is repeated. The substrate-kinase-ID is the primary key in this table, as each ID is only present once, and each combination of substrate and phosphosite is given its own ID. The foreign key in this table is substrate name, referencing the substrate table which also contains substrate name. This feature ensures that each phosphosite is actually referring to an existing substrate in the database.

The table ‘Substrate site kinase' is a relational table which maps each phosphosite-substrate instance in the ‘substrate phosphosite' table to each of the kinases it corresponds to. In this table, there are fields for a unique ID (the primary key), substrate-kinase-ID (a foreign key referencing the ID given to each phosphosite in the previous table), and a kinase corresponding to that phosphosite. Each kinase can be present multiple times in this table acting on several substrates, and each phosphosite can be acted on by multiple kinases, necessitating this table which links the kinases and substrates together. For data integrity, the kinase field in this table also has a foreign key referencing the ‘alt names' table, ensuring that the kinase that acts on the substrate, given in the data, exists.

The ‘Inhibitor' table holds the required information for inhibitors in the dataset. This information includes an assigned inhibitor ID, the name of the inhibitor, the chemical structure of the inhibitor and an image link to the structure of the inhibitor. The primary key of this table was the inhibitor ID, with no foreign key dependencies.

The relationship table ‘Kinase-inhibitor' stores information on how each inhibitor acts on kinases, and how each kinase is acted on by multiple inhibitors. In this table, there are fields for a Unique ID (primary key for the table), an inhibitor ID (a foreign key which references an inhibitor ID assigned in the ‘inhibitor' table), and a kinase which that inhibitor is known to inhibit. For integrity the kinase name given in this table is also a foreign key referencing the

kinase names in the alternative names table, ensuring that the kinase given in the inhibitor data is actually a human kinase in the database.

Phosphoproteomics Analysis

A user is able to upload quantitative phosphoproteomics experimental data in a .tsv file format to the website for data analysis. Figure 3 below shows a general overview of the inputs, the analysis process, and the outputs. The file

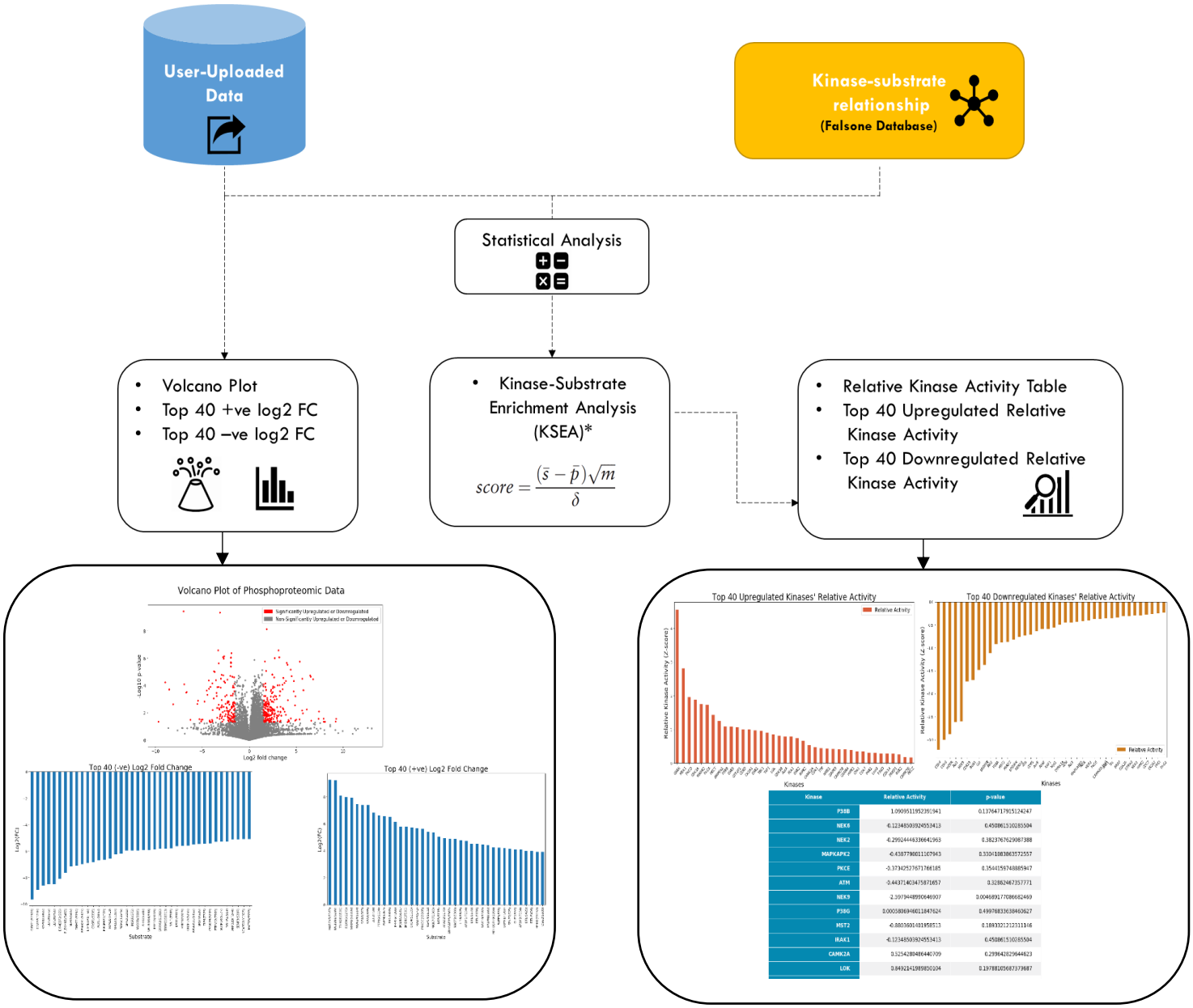

Figure 3 Depiction of the phosphoproteomics analysis process and graphical outputs.

upload requirements are that it includes one inhibitor in the data set with no more than seven columns. The analysis provides a graphical summary of the results and calculates the relative kinase activity of all human kinases represented by the dataset. A detailed explanation of the basis of the computational analysis and how it was achieved will be given, before moving on to the results interpretation.

To start with, the raw data file in .tsv format is 'cleaned-up' prior to analysis. All manipulation of the datasets was achieved using the *pandas* Python Data Analysis Library (Perez and Granger, 2007). To allow the python code to be adaptable to various data files, the header names of the raw file are changed to a fixed set of column names, so that they are consistent with header names in the python code for further data manipulation. Additionally, rows with empty cells, which are represented as 'NaN' ('not a number') are dropped from the data set. Furthermore, any cells with infinite values (Inf) as strings in the raw data file are treated as 'not a number' due to the complexity of including those in logarithmic and other mathematical operations.

Other raw data manipulation includes filtering the substrates that are not phosphorylated by a kinase at the serine (S), threonine (T), and tyrosine (Y) residues. Substrates that are not phosphorylated are removed; these are indicated by (None) and dropped from the raw data file. In addition, substrates with methionine residues are also disregarded; these are indicated by (M) in the raw data file.

Once filtration is completed, we are able to analyse the data and produce graphical summaries using matplotlib (Hunter et al., 2007). At this point, the data is graphically represented by a volcano plot of -log10 of the p-value versus log2 of the fold change of the phosphosites, which indicates upregulation or downregulation in the presence of a kinase inhibitor relative to statistical significance. The Numpy (Oliphant, 2007), scientific computing package for Python, was used to calculate the log2 fold change and -log10 p-values. In our web-application, phosphosites are deemed significantly up- or down-regulated if they have an adjusted p-value less than or equal to 0.05 with a log2 fold-change greater than 1.5, for upregulated, or less than -1.5, for downregulated.

The 'cleaned' data file is queried against our Falsone database which contains kinase-phosphosite relationships based on those that have been verified experimentally and released on PhosphoSitePlus (Phosphosite.org, 2019). This is done in order to find the kinases responsible for the phosphorylation of the phosphosites.

Once the kinases are identified for each phosphosite from the experimental data, statistical analysis is used to find the relative kinase activity based on the Kinase-Substrate Enrichment Analysis (KSEA) method (Wiredja, Koyutürk and Chance, 2017). For this method, four variables were found to use in the following formula proposed by Casado et al. (2013):

$$score = \frac{(\bar{s} - \bar{p})\sqrt{m}}{\delta}$$

where $\bar{s}$ is the average log2 fold change of the known phosphosite substrates of that given kinase in the dataset, $\bar{p}$ is the average log2 fold change of all phosphosites in the dataset, m is the total number of substrates represented by the given kinase in the dataset, and δ is the standard deviation of the log2 fold change of all the phosphosites in the data.

Mathematical calculations on those four variables, obtained from the data, allow us to build a table of the relative kinase activity, expressed in z-score, and a function from SciPy (Oliphant, 2007) library for python allows us to obtain the p-values of each z-score. In addition, two bar charts were graphed from the table to look at the top 40 positive z-scores and a second plot shows the top 40 negative z-scores. This graphic is also colour-coded to display the statistical significance of each z-score in terms of its p-value.

Front-End Development:

Front-end development of the Falsone web-application began after the back-end coding, and database build was completed. To access the database on the front-end, SQLAlchemy was used.

- **SQLAlchemy:**

In the Flask website data is retrieved from the database and presented to the application in Flask via SQLAlchemy. SQLAlchemy is an object-relational mapper that creates a queryable instance of the database within a python environment, and then takes queries written in python syntax and returns them as results objects. SQLAlchemy allows the specification of a database file, and

the language in which it is written (in this case SQLite3), and then provides an autoload option in which the table names are specified and assigned to python variables, and then the data contained in those tables is autoloaded from Metadata via SQLAlchemy, using the engine created from the instance. In this fashion, it is possible to load a database with all of its tables and the data contained within those tables relatively easily.

The types of queries that can be written in SQLAlchemy are also very user-friendly, especially for someone more comfortable with python than with SQL. Queries can be assigned to a results variable, then the session which is being queried is specified along with the table in that session that is being queried. A user can then use 'filter' or 'filter_by' options to specify what they are looking for in the table, and from which field. For example, one could query the kinase table in their session, explicitly looking for any entries that have the kinase name 'MK01_HUMAN'. The SQLAlchemy syntax would look like:

session.query(Kinase).filter_by(Kinase_name='MK01_HUMAN').all()

Moreover, the query would return all of the items with that kinase name. specified by the '.all()'. SQLAlchemy also supports the .first() and .one() extenders which would replace .all(), and return either the first instance in the table or the only instance in the table respectively. In the queries written for this software, '.one()' is often used where a query is only expected to return one result, such as when kinase name is queried.

Another majorly helpful function of SQLAlchemy which is used throughout the software is the '.like()' function of queries. With this, you can query a field in a table for instances which contain something which contains to some capacity whatever you're filtering by. To take an example from the software:

session.query(kinase).filter(kinase.c.Kinase_name.like(%search%)).all()

This query takes the kinase table again, filters explicitly the Kinase_name field of that table, and filters out anything that does not contain whatever string the user has input. If a user had input 'MK0', this query would return all of the instances in the table where a kinase name contains the string 'MK0' at any position in the name, e.g. 'MK01_HUMAN', 'MK02_HUMAN', 'AMPK01_HUMAN' etc. These .like() queries are used in the software to build the intermediate search results pages, styled after the likes of Google or UniProt, with the idea being that a user might not always know exactly what they are searching for and so should be able to input what they know and browse the results rather than having to query with an exact input. This makes the user experience much easier, and the site much more accessible for more casual users or people without a heavy background in human biology.

SQLAlchemy was a great fit for the website and the capabilities of its backend developer. It performed exactly as expected with remarkable speed and reliability, and with clear documentation/support with other modules such as Flask. There were no obvious limitations to using SQLAlchemy; it was a suitable fit for what was required.

- **Front-end program requirements:**

Below are the program requirements and relevant information on the launch of the Team Falsone website application.

- **How to run the Falsone Kinase Portal:**

1. Install *pip* and *Python v3.7* on your computer
2. Run the *requirments.txt* file by typing the following command on the terminal, a text file which was created to allow a user to install all the necessary Python modules required to use to the Falsone Kinase Portal:

   Pip install -r requirements.txt
3. Run *app.py* on the Terminal
4. Copy the generated URL on the terminal in a web browser, e.g. Firefox.

Software Architecture:

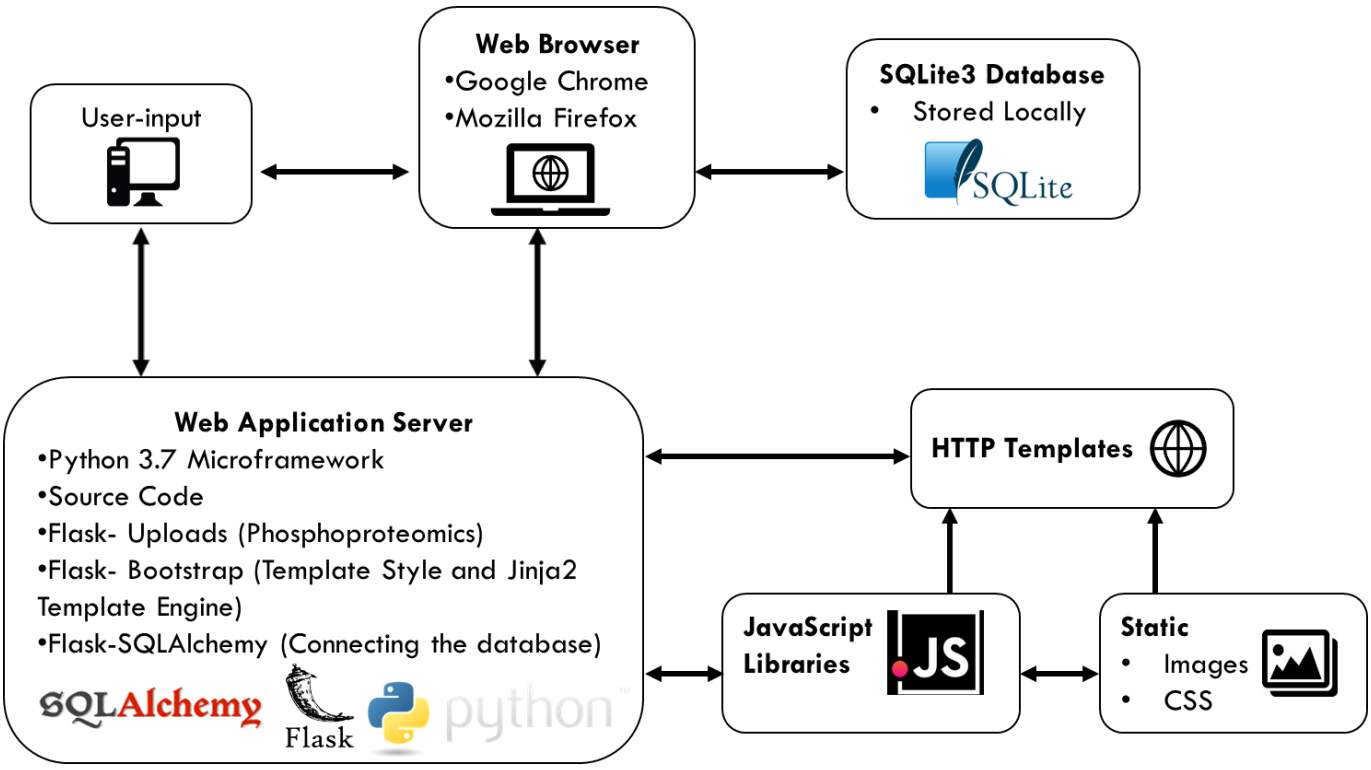

Figure 4. Schematic diagram of the Falsone web app architecture.

The Team Falsone software architecture is based on Flask, which is a micro-web framework that runs on Python version 3.7 (see Figure 4 above). Several Flask modules were used to integrate the required features of the specification. For instance, Flask-Bootstrap integrated the Bootstrap template onto the Flask application by using the Jinja2 template engine, while Flask-Uploads implemented the Phosphoproteomics feature so that the user can upload their dataset and receive customised results. Furthermore, Flask-SQLAlchemy connected the locally-stored SQLite3 database to the GUI (Graphical User Interface) of the web application. The HTML files were stored in the templates folder consisting of a parent and a child template for each page required by the Jinja2 template engine. Finally, Javascript libraries, CSS and image files were all placed in the static folder. Consequently, this arrangement of modules and files rendered the web pages viewed on the client's web browser, such as Firefox or Chrome, ready for user input.

Major technologies and tools used:

Python (version.3.7)

URL: https://www.python.org/

Python is an object-oriented open source programming language which has a simple syntax that is easy-to-implement. These features infer that it is very suitable for commercial web application development and allows for a faster development cycle, compared to more traditional programming languages, such as Java or C++.

HTML 5 (Hyper-Text-Mark-up Language):

HTML 5 is the major front-end used to display the different elements on the web page. Without HTML5, it would not be possible to render the webpage, regardless of other technologies used. Consequently, it is the major web scripting language to create a web app. Furthermore, HTML structures the web pages that are rendered in the web browser.

CSS (Cascade Style Sheet):

CSS is used to style the HTML web page. It can be used to specify fonts and adjust the layout of the web page. For this project, Bootstrap 3 CSS files were used to style the web app. Each page except the error pages has their own two CSS files, one for the style of the web page and one for the fonts.

JQuery (JavaScript) plugins v3.3.1:

URL:https://www.jqueryscript.net/table/Export-Html-Table-To-Excel-Spreadsheet-using-jQuery-table2excel.html

JQuery plugins are a group of JavaScript scripts that integrate extra functionality to the HTML and CSS code. For example, these plugins make tables searchable and enable the pagination of the tables. Therefore, these plugins were used to make the user experience of the Flask app more intuitive and seamless. Furthermore, because JQuery plugins are relatively easy to implement, hence this is why the following JQuery plugins were integrated with the web app.

JQuery DataTables v1.10.18:

URL:https://www.jqueryscript.net/table/Export-Html-Table-To-Excel-Spreadsheet-using-jQuery-table2excel.html

This JQuery plugin was used because it converts the HTML table into full functional tables with a searchable bar and pagination of the HTML tables. This plugin was used mainly for the making the kinase HTML table into a more fully functional one where the user can search for a particular kinase and its various attributes, such as accession number and gene family.

Export2excel v1.1.2:

URL:https://www.jqueryscript.net/table/Export-Html-Table-To-Excel-Spreadsheet-using-jQuery-table2excel.html

This Jquery plugin was used because it enables users of the web app to export the kinase HTML table as an xls format, ready for further data analysis on Microsoft Excel. Furthermore, the JQuery DataTables plugin made it easy to export the data queried in the table.

Bootstrap v3.0 (HTML, CSS and **JavaScript):**

Bootstrap is a front-end open source toolkit that already has pre-built HTML, CSS and JS components. This feature allows for faster development time and reduces time designing the different elements of the webpage, such as the search bar and navigation buttons. Therefore, a pre-built Bootstrap 3 template was used to design the GUI (Graphical User Interface) of the web app. This feature allowed

greater freedom to prototype with different design elements to build a better user experience.

URL: https://getbootstrap.com/

The Bootstrap template used:

Agency Bootstrap Template:

https://startbootstrap.com/themes/agency/

Flask (version 1.0.2)

URL: http://flask.pocoo.org/

Flask is a micro web framework written that is compiled in Python. It is classified as a micro-framework because it does not require external tools or libraries. This feature enables front-end web developers to combine the versatility and capabilities of Python to develop customised interactive web apps with Flask.

Flask Directory Structure

Flask folder:

The parent folder has the main Python files to run the web application. It also has other Python scripts that are defined as functions. These functions are then imported into the main *app.py* as modules. This feature was used for the Phosphoproteomics feature of the web app, where the *app.py* imports the *process_file* function *from uploadfinal.py* to generate the results.

Templates folder:

This folder has all the HTML templates that are required for the template engine called Jinja2 to find and render on the web browser. The templates are set up so that the Flask – Bootstrap module can render the web pages.

Static folder:

This folder has all the Bootstrap CSS, JS and images needed to render the templates. These files do not change under any circumstances, hence the folder name static.

Flask – Uploads (version 0.2.1):

URL: https://pythonhosted.org/Flask-Uploads/

The Flask Uploads module was used to integrate the Flask website to allow the user to upload their Phosphoproteomics dataset to generate customised graphs via Matplotlib for the user to save for further data analysis.

Flask –Bootstrap (version 3.3.7):

URL: https://pythonhosted.org/Flask-Bootstrap/

The Flask-Bootstrap uses the Jinja2 template engine to render the web pages when the Flask server runs. It uses block tags to define the different HTML elements, for example, the header, body and footer. Each block has a closing tag. The parent template lays the structure of the web page while the child template shows the body section of the web page. The Jinja2 template engine then combines both the parent and child template HTML files to render the web page.

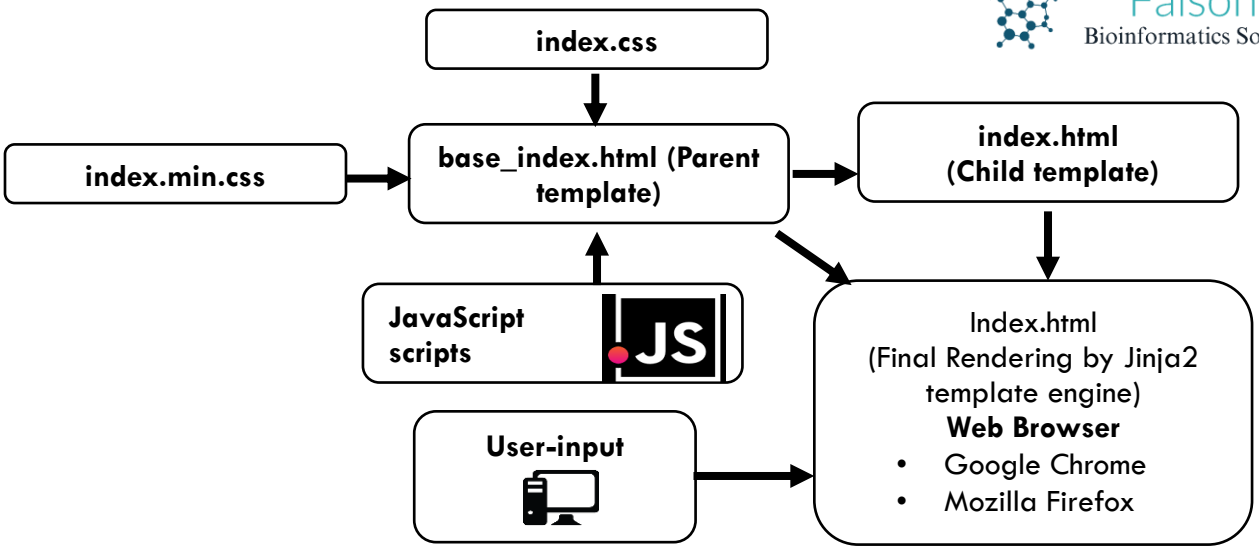

Figure 5. The Jinja2 template engine in Flask combines the parent template (CSS and Javascript libraries) and the child template together to render on the client's web browser.

Figure 5 above illustrates how the Jinja2 template engine in Flask- Bootstrap renders a web page on the Flask app. When the Flask app runs, the Jinja2 template engine first looks for the base parent template, which is linked to all the required CSS and JS files. The parent template has the basic HTML structure, but the body section is extended and linked to the child template. The parent template and child template are then rendered together on the client's web browser, e.g. Firefox.

Flask- SQLAlchemy (Version 2.3.2):

URL: http://flask-sqlalchemy.pocoo.org/2.3/

The Flask-SQLAlchemy uses the Object Relational Mapper (ORM) technique to create relationships between data (objects) between different tables. This feature creates a data mapper pattern which makes it easy to import the schema for the SQLite3 database.

Structure of a base HTML file (Parent Template):

- **{% block head %}**

  This block has the header section of the webpage. It also links the necessary CSS files and fonts.

- **{% block navbar %}**

  This block defines the navigation bar of the webpage.

- **{% block body %}**

  This block defines the body section of the webpage. In the parent template, this is extended to the child template to fill out.

- **{% block footer %}**

  This block defines the footer section of the webpage.

- **{% block script %}**

  This block contains all the JS Query scripts that add extra functionality to the web app. For example, Jquery DataTables.

Structure of an HTML file (Child Template):

- **{% extends "base_index.html" %}**

  This line is crucial because it links the parent template to the child template. This method is how the template engine connects these two files to render the web page.

- **{% block content %}**

  This block defines the content of the webpage. It is left empty because the main content of the web page goes in the body tag block.

- **{% block body %}**

This block defines the body section of the webpage. This block tag is the main content of the web page.

Using Flask Uploads in the Phosphoproteomics Analysis section:

After the user uploads a phosphoproteomics .tsv dataset file on the website, *app.py* starts Flask-Uploads and then initialises the *process_file* function in *uploadfinal.py*. Then, *uploadfinal.py* imports the .tsv file into Pandas and subsequently uses the Matplotlib module to analyse the results, generate, and redirect to the results page. The generated graphs are stored in the *static* folder, which is linked to display on the results page.

Linking the SQLite3 Database to the GUI using Flask- SQLAlchemy:

The SQLite3 database is stored locally in the Parent folder. The database is loaded onto the web app via Flask-SQLAlchemy in the main *app.py* file under both *kinase_ results* and *inhibitor_result* URL routes. The autoload function in Flask- SQLAlchemy automatically connects all the tables and the associated data onto the Flask app.

When the user searches for a kinase or an inhibitor in the search bar, it converts the search term as a string and inputs the value of the string into the query code. The query code runs, and the result is produced as a list and then redirected to the results template. The inhibitor search bar is named *search2*, while the kinase search bar is called *search* to avoid any errors. Finally, the results template is rendered.

Commonly Encountered Errors:

The most frequently encountered error throughout this project was the URL routing error which caused pages in the website to direct to the wrong web page. This technical error made the web app very unstable at certain times but a lot of time and effort was put to solve this technical issue. However, it is not entirely fixed so that the error could occur at certain times.

Technical Limitations:

For the phosphoproteomics section, when the user uploads their phosphoproteomics dataset, the graphical images generated have the same file names. This function could be a problem if many users are uploading their files at the same time. This limitation could slow down the whole Flask app, especially if the uploaded datasets are large.

Consequently, enhancements in future versions would include generating the graphical phosphoproteomics results with different file names, rather than the same file names. This feature would enable the Flask app to be used by multiple users at the same time, without the risk of it crashing.

Furthermore, because the database is stored locally, there is a risk of the database file accidentally being deleted or corrupted during data transfer. If that occurs, users would not be able to search for kinases or inhibitors, as there would be no database file available.

Accordingly, in the future, the database should be connected to an external database, such as MySQL, which is stored on the cloud. This enhancement could

reduce the loading speed and also overcome the risk of accidentally deleting or corrupting the database file, as it is stored on the cloud, rather than locally on the drive.

References:

Bootstrap. (2018). Bootstrap. Available: https://getbootstrap.com/

Casado, P., Rodriguez-Prados, J., Cosulich, S., Guichard, S., Vanhaesebroeck, B., Joel, S. and Cutillas, P. (2013). Kinase-Substrate Enrichment Analysis Provides Insights into the Heterogeneity of Signaling Pathway Activation in Leukemia Cells. Science Signaling, 6(268), pp.rs6-rs6.

David Lord et al. (2018). Flask. Available: http://flask.pocoo.org/docs/1.0/license/

Flask Bootstrap. (2018). Flask Bootstrap. Available: https://pythonhosted.org/Flask-Bootstrap/

Flask Uploads. (2018). Flask Uploads. Available: https://pythonhosted.org/Flask-Uploads/

Flask-SQLAlchemy. (2018). Flask-SQLAlchemy. Available: http://flask-sqlalchemy.pocoo.org/2.3/

Hunter, J. (2007). Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3), pp.90-95.

Kim S, Chen J, Cheng T, Gindulyte A, He J, He S, Li Q, Shoemaker BA, Thiessen PA, Yu B, Zaslavsky L, Zhang J, Bolton EE. PubChem 2019 update: improved access to chemical data. Nucleic Acids Res. 2019 Jan 8; 47(D1):D1102-1109. doi:10.1093/nar/gky1033. [PubMed PMID: 30371825]

Kinase-screen.mrc.ac.uk. (2019). Kinase Profiling Inhibitor Database | International Centre for Kinase Profiling. [online] Available at: http://www.kinase-screen.mrc.ac.uk/kinase-inhibitors [Accessed 13 Feb. 2019].

Kinase-screen.mrc.ac.uk. (2019). Kinase Profiling Inhibitor Database | International Centre for Kinase Profiling. [online] Available at: http://www.kinase-screen.mrc.ac.uk/kinase-inhibitors.

Oliphant, T. (2007). Python for Scientific Computing. Computing in Science & Engineering, 9(3), pp.10-20.

Perez, F. and Granger, B. (2007). IPython: A System for Interactive Scientific Computing. Computing in Science & Engineering, 9(3), pp.21-29.

Phosphosite.org. (2019). [online] Available at: https://www.phosphosite.org [Accessed 9 Feb. 2019].

Pubchem.ncbi.nlm.nih.gov. (2019). The PubChem Project. [online] Available at: https://pubchem.ncbi.nlm.nih.gov/

Python Software Foundation. (2018). Python. Available: https://www.python.org/. Last accessed 20/01/2019.

Uniprot.org. (2019). pkinfam.txt. [online] Available at: https://www.uniprot.org/docs/pkinfam

Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)

Wirbel, J., Cutillas, P. and Saez-Rodriguez, J. (2018). Phosphoproteomics-Based Profiling of Kinase Activities in Cancer Cells. Methods in Molecular Biology, pp.103-132.

Wiredja, D., Koyutürk, M. and Chance, M. (2017). The KSEA App: a web-based tool for kinase activity inference from quantitative phosphoproteomics. Bioinformatics, 33(21), pp.3489-3491.