BIO727P

Bioinformatics Software Development Group Project 2020

Documentation

Yutang Chen

Celia De Los Angeles Colomina Basanta

Anna Dearman

Juan Ledesma Moreno

Katie Skinner

jacky-03.ehym3crjpy.eu-west-2.elasticbeanstalk.com/home

github.com/celiaccb/Software-Development-Group-Project-2020

Contents

1. Introduction

1.1 Purpose

The aim of this document is to describe the process of developing the prototype software JACKY, how to execute it, and opportunities for future development.

1.2 Scope

Protein kinases are enzymes that modify other proteins by adding a phosphoryl group to specific amino acids. The phosphorylation of these specific sites, called phosphosites, can modify the protein's activity and lead to the alteration of biological processes in the cell.

When the activity of kinases is deregulated, the disruption of downstream pathways can cause diseases such as cancer or metabolic and inflammatory disorders. Several drugs have been tested and developed to inhibit the activity of disease-causing kinases.

JACKY is a new web-based software that unifies kinase, phosphosite, disease and inhibitor information in one convenient online platform. The software is based on a relational database containing linked tables of data. Users can also upload their experimental quantitative phosphoproteomics results to our data analysis tool to compute estimated relative kinase activity scores.

The whole package is deployed on Amazon Web Services, and thus is accessible anywhere.

1.3 Intended user and use

This software will be useful to anybody studying kinases in an academic or research setting. The application will be free of charge and open for use by everyone, with no need to register.

1.4 Version

This is version 1.0 of the JACKY software, a working prototype developed and implemented in February 2020 by students of the MSc in Bioinformatics at Queen Mary University of London, United Kingdom.

2. Requirements

2.1 Functional Requirements

The software must be able to:

I. **Have a system to retrieve information from existing online resources about kinases, phosphosites, diseases and inhibitors**
After identifying web resources, a method of raw data retrieval will be determined for each website. Original python scripts will be generated to retrieve, parse and combine the data of interest. Data will be output as csv files.

II. **Store information**
A relational database management system (RDBMS) will be used to create and handle the database in which to store the information. A schema will be devised to match the specific parameters of the csv files used in requirement I. A Python script will create the database using information in the csv files.

III. **Allow the information to be retrieved using queries**

Queries and subqueries will be written using a language that interrogates the database for the information required by the user

IV. **Allow the user to interact with the application**
The user will be able to input search terms in the web interface, and upload tsv files with quantitative phosphoproteomics data for analysis.

V. **Perform analysis with the user's data**
The software will use a python script, based on a published statistical approach, to calculate the relative activity of kinases based on user-submitted quantitative phosphoproteomics datasets. The script will calculate kinase activity using kinase-substrate data in the database**.** Several tables will be produced for the user to download:

- o Kinase(s) Relative Activity Scores (also visualised as a bar chart)
- o All phosphosites: each phosphosite from the user-submitted file, with either a matching kinase or "Not_Found" (also visualised as a volcano plot, minus unphosphorylated peptides and those with 0 values in "Fold Change" column)
- o Phosphosites with a kinase match
- o Phosphosites with no kinase match

VI. **Be accessible as a website**
The app should be deployed on Amazon Web Services so that users can access it from anywhere

2.2 Non-Functional Requirements

I. The use of the software will be totally free for the user
II. No registration will be needed for the use of the website
III. The software files will be stored in a GitHub repository
IV. The software prototype will perform as quickly, and with as few bugs, as possible.

3. Specifications

3.1 Building the relational database

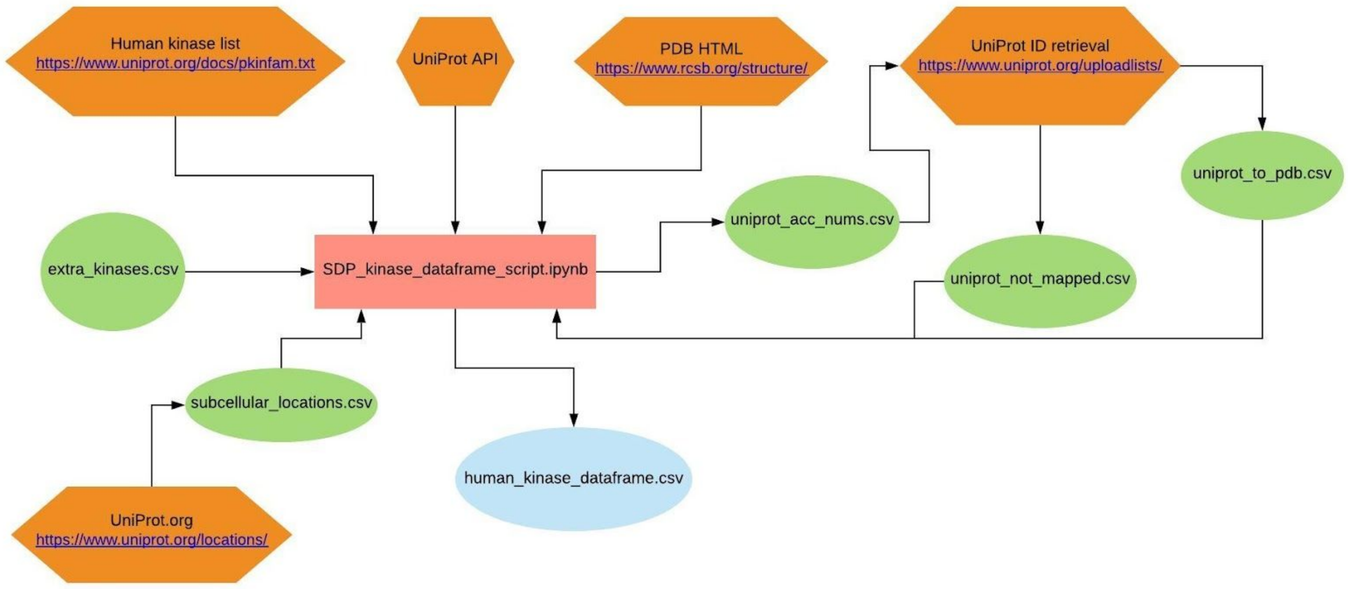
In order to feed the database, external public databases were identified and relevant data was retrieved, processed and exported as csv files. These files were then converted into a relational SQL database.

A system of queries to interrogate the relational database were created and integrated in a platform via a website framework to allow the user to interact with the data. The software also allows the user to upload their own files for data analysis.

3.1.1 Retrieve data from websites

The following steps summarise the processes to extract data from the various online repositories:

3.1.1.1 Human Kinases



Human kinase list
https://www.uniprot.org/docs/pkinfam.txt
UniProt API
PDB HTML
https://www.rcsb.org/structure/
UniProt ID retrieval
https://www.uniprot.org/uploadlists/
uniprot_to_pdb.csv
uniprot_acc_nums.csv
extra_kinases.csv
SDP_kinase_dataframe_script.ipynb
uniprot_not_mapped.csv
subcellular_locations.csv
human_kinase_dataframe.csv
UniProt.org
https://www.uniprot.org/locations/

Figure 1. This figure is a flow diagram to show the steps taken to acquire human kinase data.

To gather information on human kinases, the following sources were used:

- UniProt (https://www.uniprot.org/)
- Protein Data Bank (PDB) (https://www.rcsb.org/)

The choice to use both UniProt and PDB was an easy one. UniProt is a very reputable resource for everything protein related. Similarly, PDB is known for being a trusted protein data source. Furthemore, UniProt incorporates PDB data, and therefore using the two sources together would stand to reason.

The steps below outline the process by which the human kinase data was acquired.

1. A list of human kinases was acquired from UniProt at https://www.uniprot.org/docs/pkinfam, and extracted by opening the HTML source code of the website using a library in python called ‘urllib’.
2. Once the webpage source code was assigned to an object in python, using a regular expression, the UniProt entry names and accession numbers of the human kinases were extracted and assigned to a list object.
3. The UniProt accession IDs only were exported as a .csv file called uniprot_acc_nums.csv for later use.
4. To acquire the relevant information about each kinase, the UniProt application programming interface (API) was used to access and query UniProt data programmatically, through python. To do this, query URLs were constructed to retrieve the desired information, and extracted in the same manner as the human kinase list, using the python library ‘urllib’. Table 1 illustrates the information that was gathered and the corresponding URL that was constructed to extract that information.

Information about kinase	UniProt API query URL
Primary protein name	https://www.uniprot.org/uniprot/?query=id:**ACC/ID**&columns=protein%20names&format=tab
Alternative protein name(s)	https://www.uniprot.org/uniprot/?query=id:**ACC/ID**&columns=protein%20names&format=tab
Gene symbol	https://www.uniprot.org/uniprot/?query=id:**ACC/ID**&columns=genes(PREFERRED)&format=tab
Alternative gene name(s)	https://www.uniprot.org/uniprot/?query=id:**ACC/ID**&columns=genes(ALTERNATIVE)&format=tab

Protein kinase family	https://www.uniprot.org/uniprot/?query=id:**ACC/ID**&columns=families&format=tab
Amino acid sequence	https://www.uniprot.org/uniprot/?query=id:**ACC/ID**&columns=sequence&format=tab
Molecular mass (Da)	https://www.uniprot.org/uniprot/?query=id:**ACC/ID**&columns=mass&format=tab
Subcellular location	https://www.uniprot.org/uniprot/?query=id:**ACC/ID**&columns=comment(SUBCELLULAR%20LOCATION)&format=tab

Table 1. This table illustrates the information to be extracted from UniProt, and the corresponding UniProt API query URL constructed to acquire that data. The red 'ACC/ID' depicts where in the URL the UniProt accession ID would be inserted.

5. For each of the above URLs, once the HTML source code was retrieved and assigned as an object, different methods could be applied to extract the required information, depending on the way it was displayed.
6. Firstly, each entry extracted had a header, and to remove it, the string was split by the new line (\n) separator. Note that when a string is split in python, it becomes a list. To pull out the data, the following methods were used:
    a. Primary protein name: A regular expression was used to firstly remove all 'EC' accession IDs that were related to the kinase. The primary protein name was not contained in brackets, so by using another regular expression, any names not contained in brackets were extracted and assigned to a list.
    b. Alternative protein names: Using the same URL that was used for primary protein names was also used for alternative protein names. The aliases associated with the kinase were displayed brackets and using a regular expression, any names contained within brackets were extracted and assigned to a list object.
    c. Gene symbol: Here only one gene symbol was displayed, and thus after splitting by new line, the resulting second element of the list object could be appended to a new list.
    d. Alternative gene names: In this entry, often a few names were listed, and separated by a space. Thus, after splitting by new line, the resulting second element of the list underwent an editing, where the spaces were replaced by commas to form a user friendly readable list. Once this was done, the string was appended to a new list.
    e. Families: For family information, the string was split by new line, and as there was no undesired information to remove, the second element of the list was assigned to a new list.
    f. Amino acid sequence: Similarly for sequence information, the string was split by new line, and as there was no undesired information, the second element of the list was assigned to a new list.
    g. Molecular mass: Once again, the string was split by new line, and as there was no undesired information, the second element of the list was assigned to a new list.
    h. Subcellular location: As for subcellular location information, this string contained PubMed IDs and notes which had to be removed. For notes, they could be removed using a regular expression. However, for PubMed IDs, there was no easy way to do this via regular expressions, thus a list of all possible subcellular locations was found on UniProt at https://www.uniprot.org/locations/, downloaded (subcellular_locations.csv) and loaded in via pandas. Then using a list comprehension, if part of the string was found in the locations list, it was appended to a list and converted to a string. Then the list was appended to another list for all subcellular locations information.
7. Using pandas, a data frame was created, where each list isa column in the resulting table.
8. Next, protein structure information was required. To do this, PDB was used. Firstly, UniProt accession IDs (using the file uniprot_acc_nums.csv) were converted into PDB accession IDs using the UniProt ID retrieval system, available at: https://www.uniprot.org/uploadlists/.
9. The mapped and unmapped IDs were downloaded (uniprot_to_pdb.csv and uniprot_not_mapped.csv respectively) were then loaded back in via pandas.

[redacted]

10. UniProt ID to PDB ID mappings are one-to-many, meaning one UniProt ID sometimes has many PDB IDs associated with it. Thus, to avoid overcomplicating the dataframe, any rows with duplicate UniProt IDs were removed from the table, to result in a one-to-one UniProt ID to PDB ID mapping.
11. The mapped PDB IDs were then inserted into 2 standardised URLs as below, whereby (a) is to acquire the structure image of the kinase, and (b) was used to pull out the title of the structure:
    a. https://cdn.rcsb.org/pdb/images/**PDBID**_asym_r_500.jpg
    b. https://www.rcsb.org/structure/**PDBID**
12. The constructed image URL was appended to a list to form a column in the dataframe.
13. To extract the title of the structure, the 'urllib' library was used, again, to acquire the HTML source code of the webpage and assign it as a python object. Then a regular expression was used to parse the string. The resulting structure title was assigned to a new list to form a column in the dataframe.
14. As for the unmapped UniProt IDs, the uniprot_unmapped.csv was turned into a list and appended to the end of the mapped UniProt ID list.
15. Then, "N/A" was appended to the end of the PDB ID list and URL list.
16. Next, "No Protein Data Bank Structure Available" was appended to the end of the title list, to allow the user to know that a human structure for this protein is not currently available.
17. All the lists were then used to create a pandas data frame, with the lists becoming the columns.
18. The final step was to merge to the data frame containing the information from UniProt, with the information from PDB. The merge was done on the UniProt IDs, as both data frames contained that information, and this allowed the rows from both data frames to align correctly.
19. Penultimately, the data frame was exported as a .csv called human_kinase_dataframe.csv.
20. Finally, load in the extra_kinases.csv, identified within section 4.1.2., append the UniProt IDs and entry names to the end of the UniProt ID and entry names list, and re-run the script to contain all the kinases with their respective information.

The code for this work can be found in the script called 'SDP_kinase_dataframe_script' and is available in both .py and .ipynb format. The script was last run on the 26th of January 2020, and therefore this is when the human kinase information was last updated.

[redacted]

3.1.1.2 Phosphosites

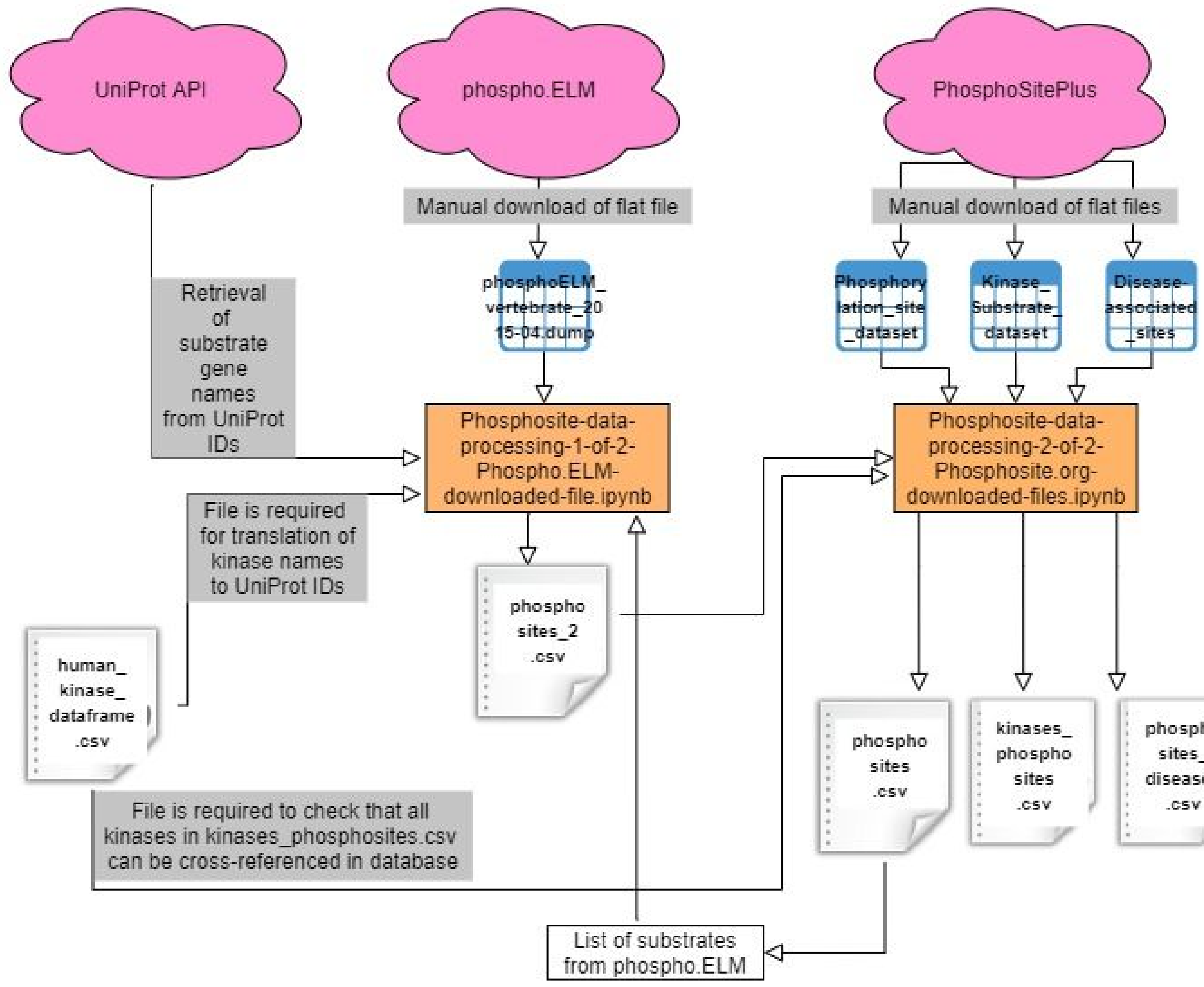

UniProt API
phospho.ELM
PhosphoSitePlus
Manual download of flat file
Manual download of flat files
phosphoELM_vertebrate_2015-04.dump
Phosphorylation_site_dataset
Kinase_Substrate_dataset
Disease-associated_sites
Retrieval of substrate gene names from UniProt IDs
Phosphosite-data-processing-1-of-2-Phospho.ELM-downloaded-file.ipynb
Phosphosite-data-processing-2-of-2-Phosphosite.org-downloaded-files.ipynb
File is required for translation of kinase names to UniProt IDs
phosphosites_2.csv
human_kinase_dataframe.csv
phosphosites.csv
kinases_phosphosites.csv
phosphosites_diseases.csv
File is required to check that all kinases in kinases_phosphosites.csv can be cross-referenced in database
List of substrates from phospho.ELM

Figure 2: Phosphosite data flow

1. Ensure you have run “SDP_kinase_dataframe_script.ipynb” to produce “human_kinase_dataframe.csv”.

2. Download the necessary phosphosite data files:

    a. Go to http://phospho.elm.eu.org/dataset.html, fill in your details and click “Accept” for access to the data files, then download the phosphosite list for vertebrates only: “phosphoELM_vertebrate_latest.dump.tgz” and unzip it to “phosphoELM_vertebrate_2015-04.dump”, or updated equivalent.

    b. Go to https://www.phosphosite.org/staticDownloads and click “I agree to abide by these conditions” for access to the data files, then download “Disease-associated_sites.gz”, “Kinase_Substrate_Dataset.gz” and “Phosphorylation_site_dataset.gz”. Unzip to get the files within, e.g. “Disease-associated_sites”, “Kinase_Substrate_Dataset” and “Phosphorylation_site_dataset”, respectively. In Notepad++ (NOT Excel), remove top three rows and save.

3. Run the Jupyter notebook “Phosphosite-data-processing-1-of-2-Phospho.ELM-downloaded-file.ipynb”. It has two inputs: “phosphoELM_vertebrate_2015-04.dump” and “human_kinase_dataframe.csv”. Ensure the file names in code cell 2 are correct. After importing the necessary packages, the script performs the following:

[redacted]

a. Creates a data frame "phosphosite_2_df" which will be edited by the script and will eventually form the output csv file
b. Removes rows about non-human substrates
c. Removes "NaN" values from kinases column and makes values uppercase
d. Defines a list of important substrate IDs to translate from UniProt ID to gene names
    - There are 5,374 unique substrate IDs in this data frame, but the majority of them will not be added to the database because the information is duplicated in the PhosphoSitePlus data. The urllib.parse and urllib.request functions that retrieve data from UniProt's API tend to "time out" when querying long lists, so it is helpful to limit the list to important substrates only. These are selected based on the assessment of the "phosphosites.csv" and "kinases-phosphosites.csv" files generated in an initial run-through of the scripts. UniProt IDs in rows originating from phospho.ELM that have missing gene IDs are pasted into this substrates list
e. Retrieves the substrate gene names from UniProt's API, using the list of important substrates
    - Sometimes several attempts at running this loop required, due to their tendency to "time out". If necessary, additional cells can be inserted and multiple UniProt-querying loops can be run for smaller sub-lists of substrates. The results can then be concatenated.
    - A cell is included which assigns data obtained from a successful run of this loop on 27-28 January 2020 to a list. In order to save time when testing the script, this cell can be un-commented and run.
f. Creates a dictionary of these aliases and uses them to create and attach a new column "SUB_GENE" to the data frame.
g. Translates the kinase IDs using "human_kinase_dataframe.csv"
    - The kinases in phosphosite_2_df are gene names in varying formats. The script translates as many as possible to unambiguous UniProt IDs.
    - Many kinases are translated by looking for an exact match between the string and either "Gene_Symbol" or "Entry name" ("_HUMAN" must be appended to the string first).
    - Other kinases are then translated by searching for the substring's occurrence in "Primary_Protein_Name", "Alternative_Protein_Name(s)" and "Alternative_Gene_Name(s)".
    - Other kinases have to be translated by splitting them, by a common non-alphanumeric character, into two sub-strings and interrogating "Primary_Protein_Name", "Alternative_Protein_Name(s)" and "Entry_name" for these sub-strings.
    - Some of the loops used to perform these checks do not fully complete their jobs unless run multiple times, hence the loops are included multiple times in the script. Further information is provided in the script comments.
    - It is important to manually sense-check the dictionary "kinase_dict" and modify the script as required. For example, kinase "ABL" is prevented from entering the loop that checks "Primary_Protein_Name" because it matches with the word "probable".
    - Several kinases in the original data frame cannot be translated unambiguously and have to be omitted, unfortunately.
h. Creates PHOS_ID as foreign key
    - In order to unambiguously identify each phosphosite in this table, and in the table we will create from PhosphoSitePlus data, create an ID by concatenating the UniProt ID and the amino acid residue information in parentheses, e.g. P16401(T4)
i. Removes unnecessary columns "source" and "entry_date"
j. Outputs the table in the form of a file named "phosphosites_2.csv"

4. Next, run the Jupyter notebook "Phosphosite-data-processing-2-of-2-Phosphosite.org-downloaded-files.ipynb". It has four initial inputs:

[redacted]

5. "Phosphorylation_site_dataset", "Kinase_Substrate_Dataset", "Disease-associated_sites" and "phosphosites_2.csv". Ensure the file names in code cells 2-5 are correct. Later, it also takes "human_kinase_dataframe.csv" as an input. Check code cell 35 to ensure the file name is correct. After

   importing the necessary packages, the script performs the following:
   a. Creates data frames from the four inputs, respectively: "phosphosite_df", "kin_sub_df" and "disease_df" which will be edited by the script and eventually form the output csv files, and "phosphosites_2_df", from which rows will eventually be copied to phosphosite_df and kin_sub_df.
   b. Removes rows
      - about non-human kinases and substrates (all data frames)
      - for "disease_df", where "DISEASE", is blank
      - where the post-translational modification is not phosphorylation, as indicated by "-p" in the residue ("phosphosite_df", "disease_df")
   c. Adds a "source" column containing links to the PhosphoSitePlus page for the substrate protein ("phosphosite_df", "kin_sub_df")
   d. Adds an empty "PMID" column so that the PubMed IDs from phosphosite_2_df can be added later ("phosphosite_df", "kin_sub_df")
   e. Creates PHOS_ID5 column as foreign key ("phosphosite_df", "kin_sub_df")
      - In order to unambiguously identify each phosphosite in these tables, creates an ID by concatenating the UniProt ID and the amino acid residue information in parentheses, e.g. P05198(S52)
      - This is also added to "disease_df" as "PHOS_ID"
   f. Creates temporary KIN_PHOS_ID column to allow unique kinase-substrate relationships to be identified ("kin_sub_df", "phosphosites_2_df")
      - Creates an ID by concatenating the kinase UniProt ID, the substrate UniProt ID and the amino acid residue information in parentheses, e.g. Q9BQI3_P05198(S52)
   g. Adds rows from some tables to others, cross-checking the values in KIN_PHOS_ID, PHOS_ID and PHOS_ID5 as appropriate

Source table	Destination table
kin_sub_df	phosphosite_df
disease_df	phosphosite_df
phosphosites_2_df	kin_sub_df
phosphosites_2_df	phosphosite_df

Table 2: Phosphosite tables combined

   h. Adds four other phosphosite ID columns, using gene and protein name information rather than UniProt ID, for potential use in the data analysis functionality of the website ("kin_sub_df", "phosphosite_df")
   i. Makes gene entries uppercase ("kin_sub_df", "phosphosite_df")
   j. Removes "-p" indicator from MOD_RSD column ("phosphosite_df")
   k. Removes unnecessary columns:
      - phosphosite_df: 'ORGANISM'
      - kin_sub_df: 'KIN_ORGANISM', 'SUB_ORGANISM', 'KINASE', 'SUBSTRATE', 'SUB_GENE_ID', 'SUB_MOD_RSD', 'SUB_GENE', 'SITE_GRP_ID', 'SITE_+/-7_AA', 'DOMAIN', 'KIN_PHOS_ID'
      - disease_df: 'PROTEIN', 'GENE_ID', 'ORGANISM', 'GENE', 'HU_CHR_LOC', 'MW_kD', 'SITE_GRP_ID', 'MOD_RSD', 'DOMAIN', 'SITE_+/-7_AA'
   l. Adds a column ISOFORM to the phosphosite_df table to indicate whether the row is for an isoform (1) or not (0)

m. Makes a new kinase accession ID column in kin_sub_df, KIN_ACC_ID_2, with any isoform characters (e.g. "-2" suffix) removed, to act as a foreign key
n. Imports “human_kinase_dataframe.csv”, and removes any rows from kin_sub_df whose KIN_ACC_ID_2 cannot be found therein
o. Removes duplicate rows from phosphosite_df based on “PHOS_ID5”
p. Creates unique identifiers for each row in each table for use as primary keys
q. Exports the three tables as csv files

3.1.1.3 Inhibitors

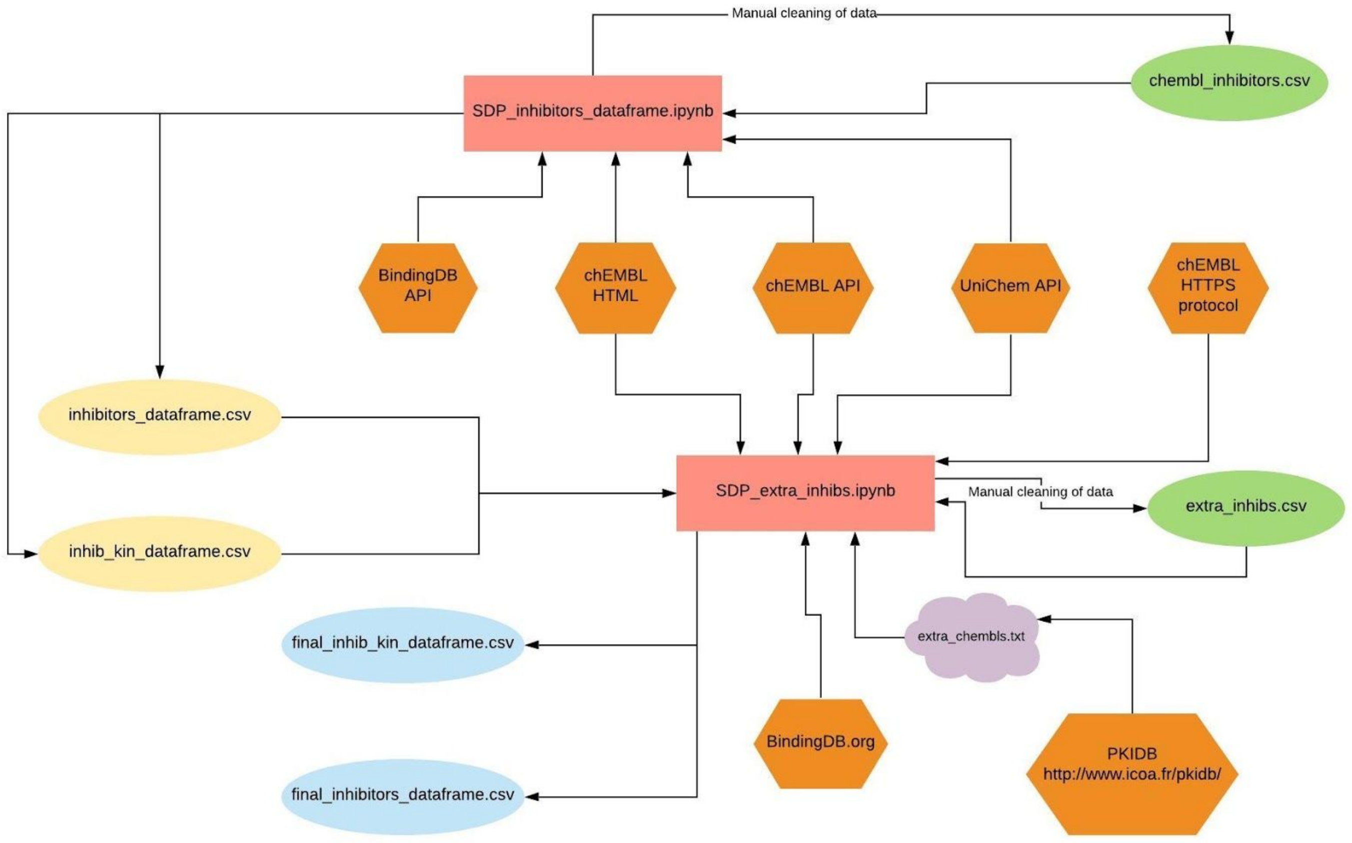


Figure 3. This figure is a flow diagram to show the steps taken to acquire human kinase inhibitors data.

To gather information on human kinase inhibitors, the following sources were used:

- BindingDB (http://www.bindingdb.org/)
- chEMBL (https://www.ebi.ac.uk/chembl/)

The choice to use BindingDB and chEMBL was a carefully considered one. BindingDB states on their website that they are “focusing chiefly on the interactions of proteins considered to be drug-targets with small, drug-like molecules”. Furthermore BindingDB contains a large amount of data: 1,819,720 binding data, for 7,470 protein targets and 804,949 small molecules. Similarly, chEMBL is a database, which is a branch of the European Bioinformatics Institute (EMBL-EBI), is defined as “a manually curated database of bioactive molecules with drug-like properties”. Moreover, the chEMBL database contains 11,000 drugs and 1.9 million compounds for 12,000 targets.

Other databases that were considered were PKIDB (http://www.icoa.fr/pkidb/) and Kinase Profiling Inhibitor Database (http://www.kinase-screen.mrc.ac.uk/kinase-inhibitors), however they only contained 218 and 243 inhibitors, respectively. Therefore, it was possible by using these resources we wouldn’t have gathered the wide range of kinase inhibitors available.

The steps below outline the process by which the human kinase inhibitor data was acquired.

1. The starting point was to import the uniprot_acc_nums.csv file acquired in the generation of the human_kinase_dataframe.csv. This file contains all the UniProt accession IDs of the human kinases.
2. Using the BindingDB API system, it was possible to search for all the molecules that were associated with the kinase by subbing in the UniProt accession ID in the URL below, where it says “ACC/ID”:
   a. http://www.bindingdb.org/axis2/services/BDBService/getLigandsByUniprots?uniprot=**ACC/ID**&&code=0&response=json
3. The data was retrieved using the library ‘urllib’ in python to get and assign the HTML source code of the entry page.
4. Then, using a regular expression, the following information about each molecule was extracted, and appended to its designated list:
   a. Monomer ID, an ID for the molecule that is linked to the BindingDB ID.
   b. Using the monomer ID it was possible to create the BindingDB ID by appending “BDBM” in front of the monomer ID.
   c. Ki, the inhibitor constant of the inhibitor.
   d. IC50, the concentration of the inhibitor where the response/binding is reduced by half.
   e. Kd, the dissociation constant of the inhibitor.
   f. EC50, the concentration of a drug that gives half maximal response.
5. Alongside the above information, the UniProt ID was appended to a list with every iteration of the regular expression parsing the HTML source code.
6. To gather more information on each inhibitor, chEMBL will need to be used. To do this, we made use of the UniChem API function, whereby it is possible to map the BindingDB IDs to chEMBL IDs, using the query URL below, whereby MONID is the monomer ID extracted from BindingDB earlier:
   a. https://www.ebi.ac.uk/unichem/rest/src_compound_id/MONID/31
7. To acquire this information, the ‘urllib’ library in python was used to parse the HTML source code, and using a regular expression, the chEMBL IDs could be extracted and appended to its designated list.
8. Once the BindingDB IDs were mapped to chEMBL IDs, a data frame was constructed with the above information gathered from BindingDB, the BindingDB ID, UniProt accession ID and chEMBL ID.
9. As some BindingDB IDs were unable to be mapped to chEMBL IDs, an “N/A” was appended. The decision was made to drop those BindingDBs that couldn’t be mapped to chEMBL for several reasons:
   a. It would be difficult to acquire the next load of data for those inhibitors, as they would either have to be retrieved from another source or entered manually.
   b. Upon inspection of some of the BindingDB entries, it is evident that these molecules do not have any commercial names given to them, and thus perhaps they were not yet in the research phase or being considered for research in the medical field quite yet, and these were most commonly those entries whereby the BindingDB ID couldn’t be mapped to the chEMBL ID. Therefore, it was sensible to exclude them for now.
10. Once the “N/A” entries were dropped, the table was simplified further by dropping duplicates of the chEMBL IDs and assigning this to a new data frame. This simplified list of chEMBL IDs acts as the basis of the inhibitors table, whereby this table contains one entry per inhibitor.
11. Next, a template URL is constructed, as below, to acquire information on each inhibitor, whereby the chEMBL ID was subbed into CHEMBLID:
    a. https://www.ebi.ac.uk/chembl/compound_report_card/**CHEMBLID**/
12. Some chEMBL entries tend to have the chEMBL accession ID as their name, and no other names listed. However, some entries have the chEMBL ID listed as the main compound name, but then have additional synonyms. Therefore, when extracting the data, a regular expression first checked if a synonyms list was present in the source code. If there wasn’t, then the entry would be skipped and no information extracted.
13. However, if a synonyms list was present, then the loop would continue, extracting the following using regular expressions, and appending them to their designated lists:
    a. Molecule name
    b. Molecule type

[redacted]

    c. Molecular formula
    d. Molecular weight
    e. Synonyms

14. At this point, it was evident that some of the molecule names and synonyms had been extracted in a strange way, including characters such as '#' and other unwanted punctuation. So all the lists created were made into a pandas data frame, and then exported as a .csv, called chEMBL_inhibitors.csv, for manual clean up in Excel, making heavy use of the 'search and replace' function.
15. Once the data was manually cleaned, the table was imported back in with pandas.
16. Then, the BindingDB data frame was merged, on the chEMBL_ID column, with the newly imported chEMBL data frame, to form the inhibitors data frame.
17. The chEMBL_IDs list was also used to create URLs for the images of each inhibitor, using the chEMBL API system, whereby the chEMBL_ID was subbed into where it says CHEMBLID below:
    a. https://www.ebi.ac.uk/chembl/api/data/image/**CHEMBLID**.png?bgColor=white
18. This list was appended to the inhibitors data frame also.
19. Finally, for the inhibitors data frame, unique accession IDs were created using a loop to add a number from the 0 to the end of the list to the string "IN" for inhibitor.
20. For the inhibitors-kinase relationships table, there was already a table whereby the human kinases, depicted as UniProt accession IDs, were mapped to molecules in BindingDB via the BindingDB ID. To complete this table, a dictionary type loop was formed, whereby the inhibitors bindingDB_ID, chEMBL_ID and molecule_name columns were extracted as lists.
21. Then, the loop would take a bindingDB_ID from the mapped UniProt ID to Binding DB ID list, and find it's index in the inhibitors bindingDB_ID. Using this index, the chEMBL_ID and molecule name_name are acquired and then appended to two new lists to form the two columns of the inhibitors-kinases relationships table.
22. Then, similarly to Step 19, unique accession numbers were created from 0 to the end of the list and added to a string "IK" for inhibitor-kinase.
23. Penultimately, 3 inhibitors were added manually to the end of the data frame as they were known to exist but were excluded from the data frame due to not conforming to the conditions in Steps 12 and 13.
24. Lastly, the two data frames were exported as inhibitors_dataframe.csv, and inhib_kin_dataframe.csv.

Additional steps to add extra inhibitors not found in the original data frame.

1. By doing some testing of the database, it was evident that there were some inhibitors missing. The chEMBL IDs of these missing inhibitors were compiled into the .txt file called extra_chembls.txt.
2. To acquire the UniProt IDs of the target human kinases these inhibitors act upon, the library 'chembl_webresource_client' in python was used. This library essentially uses HTTPS protocol to acquire the information desired.
3. Here, the chEMBL compound IDs were mapped to their targets in terms of chEMBL target IDs. Then the chEMBL target IDs were converted into UniProt IDs. This creates the basis for another inhibitors-kinases relationship table, but in this case, it is UniProt IDs mapped to chEMBL IDs.
4. The original human kinase UniProt accession ID list from Step 1 was loaded in, and through a loop, if the new UniProt ID list wasn't in the original UniProt ID list, then the index of that ID would be appended to a list. Using this 'unwanted' list of indexes, and using pandas, those targets that weren't human kinses were removed.
5. The next steps were carried out as described above, but in Step 6, instead of converting the BindingDB IDs into chEML IDs, it was reversed.
6. To acquire the BindingDB information on the 33 extra kinases, this was done manually, as it wasn't evident that the BindingDB API system supported the searching on a monomer to extract the information required. For a dataset larger than this, an alternate method would need to be devised.
7. For the rest of the data acquisition, the same protocol was used, from Step 14 to Step 24, with some minor changes, whereby in Steps 20 and 21, it was chEMBL IDs being used to find the indexes of BindingDB_IDs and

molecule names instead. Furthermore, in Steps 19 and 22, the unique accession ID generation would start from the last accession ID in each respective table.

8. Lastly, the inhibitors_dataframe.csv and inhib_kin_dataframe.csv would be loaded in via pandas, and then the extra inhibitors data frames would be merged with the main data frames to create two final data frames: final_inhibitor_dataframe.csv and final_inhib_kin_dataframe.csv.

The code for this work can be found in the scripts called 'SDP_inhibitors_dataframe' and 'SDP_extra_inhibs' and are both available in .py and .ipynb format. The scripts were last run on the 10th and 11th of February 2020, respectively, and therefore this is when the inhibitors information was last updated.

3.1.2 Creating the database

3.1.2.1 Database tools

The system of choice to handle the data is RDBMS. The advantage of using a relational database system is the easy way of managing huge amount of information in tables related one to another in a single file rather than using multiple independent files to organise related information.

SQLite (https://www.sqlite.org/) is the engine that JACKY uses as RDBMS. It does not require special configuration for its use, and interacts directly with files stored in the computer without the need for a server. Furthermore, it is free to use for any purpose. Also it uses structure query language (SQL), which offers a very simple structure to querying and handle the database.

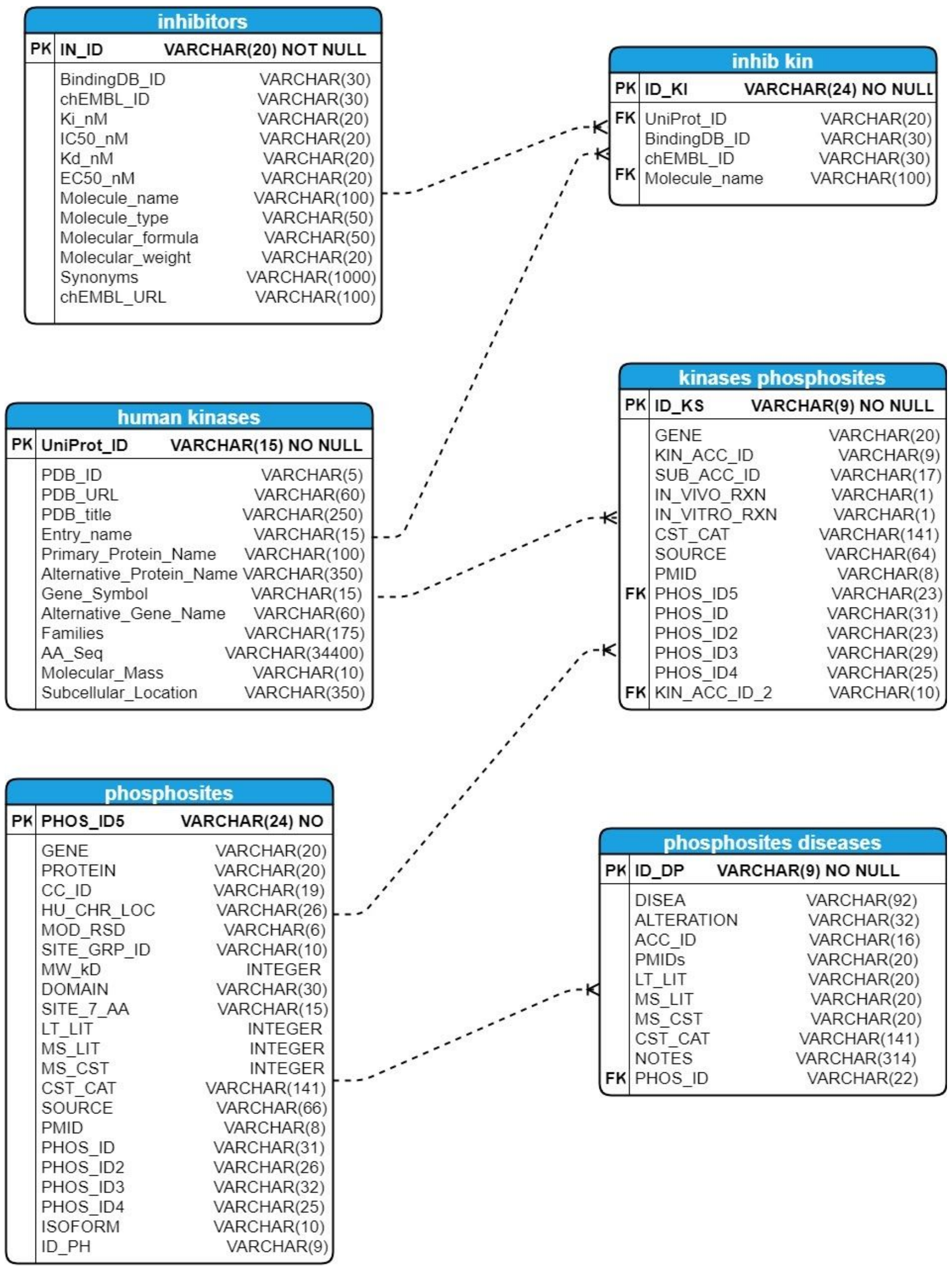
SQLalchemy (https://www.sqlalchemy.org/) is the tool to create the schema of the database to work with. This tool provides the Pythonic language to interact with SQL databases and its versatility allows to connect the queries to the database to a website framework used for the development of the application (see section 5.5).

3.1.2.2 Database schema

The schema of the relational database consists of 6 tables, defined as object or classes in accordance to the structure of SQLalchemy (see "kinases_schema.py" on https://github.com/celiaccb/Software-Development-Group-Project-2020/tree/master/database_schema). Each table contains several columns with the definition for the headers for each column and the specific features of the records to be entered in each column (Strings, integers, relationship).

Three of these tables (human_kinases, inhibitors and phosphosites) corresponds to the parent tables which Primary Keys are connected to Foreign keys on the children tables (kinases_phosphosite, inhib_kin and phosphosites_diseases, Figure 4). The relationships between each parent table and its child table are of the type one-to many.

[redacted]



inhibitors

PK	IN_ID	VARCHAR(20) NOT NULL
	BindingDB_ID	VARCHAR(30)
	chEMBL_ID	VARCHAR(30)
	Ki_nM	VARCHAR(20)
	IC50_nM	VARCHAR(20)
	Kd_nM	VARCHAR(20)
	EC50_nM	VARCHAR(20)
	Molecule_name	VARCHAR(100)
	Molecule_type	VARCHAR(50)
	Molecular_formula	VARCHAR(50)
	Molecular_weight	VARCHAR(20)
	Synonyms	VARCHAR(1000)
	chEMBL_URL	VARCHAR(100)

inhib kin

PK	ID_KI	VARCHAR(24) NO NULL
FK	UniProt_ID	VARCHAR(20)
	BindingDB_ID	VARCHAR(30)
	chEMBL_ID	VARCHAR(30)
FK	Molecule_name	VARCHAR(100)

human kinases

PK	UniProt_ID	VARCHAR(15) NO NULL
	PDB_ID	VARCHAR(5)
	PDB_URL	VARCHAR(60)
	PDB_title	VARCHAR(250)
	Entry_name	VARCHAR(15)
	Primary_Protein_Name	VARCHAR(100)
	Alternative_Protein_Name	VARCHAR(350)
	Gene_Symbol	VARCHAR(15)
	Alternative_Gene_Name	VARCHAR(60)
	Families	VARCHAR(175)
	AA_Seq	VARCHAR(34400)
	Molecular_Mass	VARCHAR(10)
	Subcellular_Location	VARCHAR(350)

kinases phosphosites

PK	ID_KS	VARCHAR(9) NO NULL
	GENE	VARCHAR(20)
	KIN_ACC_ID	VARCHAR(9)
	SUB_ACC_ID	VARCHAR(17)
	IN_VIVO_RXN	VARCHAR(1)
	IN_VITRO_RXN	VARCHAR(1)
	CST_CAT	VARCHAR(141)
	SOURCE	VARCHAR(64)
	PMID	VARCHAR(8)
FK	PHOS_ID5	VARCHAR(23)
	PHOS_ID	VARCHAR(31)
	PHOS_ID2	VARCHAR(23)
	PHOS_ID3	VARCHAR(29)
	PHOS_ID4	VARCHAR(25)
FK	KIN_ACC_ID_2	VARCHAR(10)

phosphosites

PK	PHOS_ID5	VARCHAR(24) NO
	GENE	VARCHAR(20)
	PROTEIN	VARCHAR(20)
	CC_ID	VARCHAR(19)
	HU_CHR_LOC	VARCHAR(26)
	MOD_RSD	VARCHAR(6)
	SITE_GRP_ID	VARCHAR(10)
	MW_kD	INTEGER
	DOMAIN	VARCHAR(30)
	SITE_7_AA	VARCHAR(15)
	LT_LIT	INTEGER
	MS_LIT	INTEGER
	MS_CST	INTEGER
	CST_CAT	VARCHAR(141)
	SOURCE	VARCHAR(66)
	PMID	VARCHAR(8)
	PHOS_ID	VARCHAR(31)
	PHOS_ID2	VARCHAR(26)
	PHOS_ID3	VARCHAR(32)
	PHOS_ID4	VARCHAR(25)
	ISOFORM	VARCHAR(10)
	ID_PH	VARCHAR(9)

phosphosites diseases

PK	ID_DP	VARCHAR(9) NO NULL
	DISEA	VARCHAR(92)
	ALTERATION	VARCHAR(32)
	ACC_ID	VARCHAR(16)
	PMIDs	VARCHAR(20)
	LT_LIT	VARCHAR(20)
	MS_LIT	VARCHAR(20)
	MS_CST	VARCHAR(20)
	CST_CAT	VARCHAR(141)
	NOTES	VARCHAR(314)
FK	PHOS_ID	VARCHAR(22)

Figure 4. Entity Relationship (ER) Diagram displaying the model for the tables used in the database. PK and FK correspond to primary and foreign key respectively.

3.1.2.3 Transfer of data from csv files into database

The following steps were taken to populate the database:

1. The schemas as seen in Figure 4 above were set up, followed by the definitions of the relationships between them.
2. Then the six .csv files which contain all the data were loaded in. These .csv files are:
    a. human_kinase_dataframe.csv
    b. phosphosites.csv
    c. final_inhibitors_dataframe.csv
    d. kinases_phosphosites.csv
    e. phosphosites_diseases.csv

[redacted]

f. final_inhib_kin_dataframe.csv

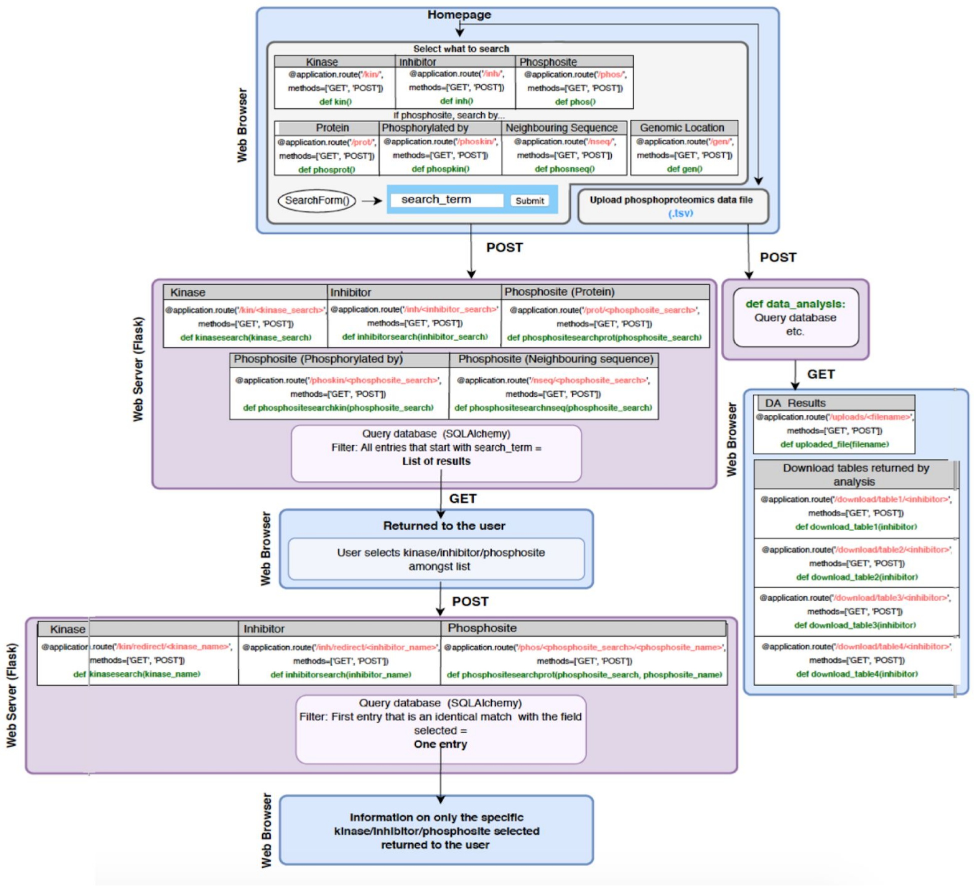
3. Once the .csv files were loaded in, loops were created to insert the correct information from the relevant .csv files into the correct columns of the database schema.
4. To do this, the .iloc[a, b] function from pandas was utilised, with 'a' corresponding to each individual row, and 'b' being equal to each column in the data frame. Therefore each piece of information was entered from the .csv file into the correct column of the table within the schema.

The code for this work can be found in the script called 'SDP_populate_db_script' and is available in both .py and .ipynb format.

3.2 Website framework

3.2.1 What is Flask & Basic Functions

Flask is a Web Server Gateway Interface (WSGI) web application framework written in Python, authored by Armin Ronacher. It is considered a micro-framework since it only has three dependencies; Werkzeug, which supports the routing, debugging, and WSGI subsystems, Jinja2, a template engine, and Click, which handles the command-line integration. All other high-level tasks can be achieved through extensions, allowing the developer to modulate the functionality of the website to best suit the user needs by choosing which extensions to include. Furthermore, its minimalistic structure makes the code easy to follow for anyone versed on Python, and it supports all relational databases.
By default, Flask looks for HTML templates (which dictate the structure of the website returned to the user) on a templates folder inside the main application folder, while it looks for static files such as images, JavaScript source files, and CSS files (which are used to refine the appearance of HTML templates) on a static folder within the main application folder.


Homepage
Web Browser
Select what to search
Kinase
Inhibitor
Phosphosite
If phosphosite, search by...
Protein
Phosphorylated by
Neighbouring Sequence
Genomic Location
SearchForm()
search_term
Submit
Upload phosphoproteomics data file (.tsv)
POST
POST
Web Server (Flask)
Kinase
Inhibitor
Phosphosite (Protein)
Phosphosite (Phosphorylated by)
Phosphosite (Neighbouring sequence)
Query database (SQLAlchemy) Filter: All entries that start with search_term = List of results
def data_analysis: Query database etc.
GET
Web Browser
DA Results
Download tables returned by analysis
GET
Web Browser
Returned to the user
User selects kinase/inhibitor/phosphosite amongst list
POST
Web Server (Flask)
Kinase
Inhibitor
Phosphosite
Query database (SQLAlchemy) Filter: First entry that is an identical match with the field selected = One entry
Web Browser
Information on only the specific kinase/inhibitor/phosphosite selected returned to the user

```
@application.route('/kin/', methods=['GET', 'POST'])
def kin()
@application.route('/inh/', methods=['GET', 'POST'])
def inh()
@application.route('/phos/', methods=['GET', 'POST'])
def phos()
@application.route('/prot/', methods=['GET', 'POST'])
def phosprot()
@application.route('/phoskin/', methods=['GET', 'POST'])
def phospkin()
@application.route('/nseq/', methods=['GET', 'POST'])
def phosnseq()
@application.route('/gen/', methods=['GET', 'POST'])
def gen()

@application.route('/kin/<kinase_search>', methods=['GET', 'POST'])
def kinasesearch(kinase_search)
@application.route('/inh/<inhibitor_search>' methods=['GET', 'POST'])
def inhibitorsearch(inhibitor_search)
@application.route('/prot/<phosphosite_search>', methods=['GET', 'POST'])
def phosphositesearchprot(phosphosite_search)
@application.route('/phoskin/<phosphosite_search>', methods=['GET', 'POST'])
def phosphositesearchkin(phosphosite_search)
@application.route('/nseq/<phosphosite_search>', methods=['GET', 'POST'])
def phosphositesearchnseq(phosphosite_search)

@application.route('/uploads/<filename>', methods=['GET', 'POST'])
def uploaded_file(filename)
@application.route('/download/table1/<inhibitor>', methods=['GET', 'POST'])
def download_table1(inhibitor)
@application.route('/download/table2/<inhibitor>', methods=['GET', 'POST'])
def download_table2(inhibitor)
@application.route('/download/table3/<inhibitor>', methods=['GET', 'POST'])
def download_table3(inhibitor)
@application.route('/download/table4/<inhibitor>', methods=['GET', 'POST'])
def download_table4(inhibitor)

@application.route('/kin/redirect/<kinase_name>', methods=['GET', 'POST'])
def kinasesearch(kinase_name)
@application.route('/inh/redirect/<inhibitor_name>', methods=['GET', 'POST'])
def inhibitorsearch(inhibitor_name)
@application.route('/phos/<phosphosite_search>/<phosphosite_name>', methods=['GET', 'POST'])
def phosphositesearchprot(phosphosite_search, phosphosite_name)
```

Figure 5. Structure and functionality of the JACKY Flask application script (application.py)

[redacted]

3.2.2 Processing user searches: Flask-WTF web forms

In the homepage of the website, you will notice a select dropdown menu in which users can select whether they want to search a kinase, phosphosite or an inhibitor. As you can see on the upper box of Fig4, when the user selects one of these options they're redirected to a different URL depending on the field selected, and for each of these URLs a route instance along with a view function is defined on the main application script. As so, when the user enters a search, the term entered will be used to make a different query to the database and return a different table of hits (see point 3.2.3).
A web form was used to process user searches, as they can be used in HTML templates to allow users to enter information, which is then submitted to the web server from the web browser as a POST request. You can integrate a web form to your Flask application script using the Flask-WTF extension, which gives you access to the forms validation and rendering Python library WTForms.
Flask-WTF requires you to set a secret key through application_instance_name.config['SECRET_KEY']
to block cross-site request forgery (CSRF) attacks. The web form is set on the Python script by writing a class that inherits from the class FlaskForm, which contains a list of fields or class variables, as in the SearchForm you can find on our Flask application script:

```
from flask_wtf import FlaskForm
from wtforms import StringField, SelectField, SubmitField
from wtforms.validators import DataRequired
class SearchForm(FlaskForm):
    protein_name = StringField("", validators=[DataRequired()])
    submit = SubmitField('Submit')
```

The class variable protein_name is a text field, as specified by StringField, to which a validator is attached, which in this case checks that data has been entered by the user, while submit is a form submission button, as specified by SubmitField.

The form can be handled on the view functions through form.validate_on_submit(), as in the next route, which handles kinases searches:

```
@application.route('/kin/', methods=['GET', 'POST'])
def kin():
    form = SearchForm()
    protein_name = None
    if form.validate_on_submit():
        protein_name = form.protein_name.data
        return redirect(url_for('kinasesearch', kinase_search=protein_name))
    return render_template('homepagekin.html', form=form)
```

The form handling is a POST request, as information is sent from the web browser to the web server.
By default, the route instances only handle GET requests, that is requests which send information from the web server to the web browser, therefore the methods argument needs to be added to a view function to specify that it should handle both GET and POST requests.
The protein_name variable stores the information received from the form when available, when it is not known it is initialized to None. The function creates an instance of the SearchForm class, which can be called with form.
If no form has been submitted, validate_on_submit() returns False, therefore the if statement is skipped, returning the homepage HTML template in which the form is rendered. If a form is submitted, the url for the kinasesearch function is returned to the user, and the term entered by the user on the form (protein_name) is used as an argument on the function to query the database, as you can see on the next point.

3.2.3 Connecting the application to a database & making queries with SQLAlchemy

[redacted]

SQLAlchemy, the Python SQL toolkit and Object Relational Mapper, was used to define & query our database.
The Flask application is connected to a database (note that the database needs to be on the main Application folder, or in a subdirectory within it) using the create_engine () instance such as:

```
from sqlalchemy import create_engine
engine = create_engine ('sqlite:///pathto/yourdatabase.db’, echo = True, connect_args=. {'check_same_thread':
False})
```

By default, SQLite prohibits the use of a single connection in more than one thread, ergo why this option had to be set to ‘False’.
The structure or schema of the database is specified on the script, first by creating a base class through the declarative_base instance, from which the entities (i.e. the database tables) are going to inherit. After defining the tables and columns of the database in the inherited classes (e.g. class HumanKinases ()), and the relationships between the tables, a call to the metadata of the Base class (Base.metadata.create_all(engine)) generates the schema. Please see point 3.1.2.2. and Figure 4 to know more about the database schema and functioning.
To query the database, a Session instance needs to be set using sessionmaker, and its connection to the database specified, as in:

```
from sqlalchemy.orm import sessionmaker
Session = sessionmaker (bind = engine)
session = Session ()
```

The database is queried to meet three requirements.
First, to return a table of hits when the user searches a term (from which they can select a specific kinase/phosphosite/inhibitor). These queries need to search the database table containing kinase/phosphosite/inhibitor information for the rows in which the column containing the kinase/phosphosite/inhibitor names contains the term searched by the user.

For example, this is the query to search a kinase:

```
results=session.query(HumanKinases).filter(or_(HumanKinases.Entry_name.startswith(kinase_search),
HumanKinases.Gene_Symbol.startswith(kinase_search)) ).all()
```

The Entry_name column on the HumanKinases table contains the names of the kinases on our database. Additionally, it is common for biologists to search kinases by its gene name, which in the HumanKinases table is present on the Gene_Symbol column, therefore the query isolates the rows on the HumanKinase table in which the entry on the Entry_name column or the Gene_Symbol column starts with kinase_search, and returns all results as a list. The information can be accessed through queryname.columname, for example in this case, results.Entry_name would return only the kinase name of all the entries present on the list.

Secondly, to return a page containing information specific for the kinase/phosphosite/inhibitor the user has selected from the hits page. After the user searches for a term and the website returns a table with all the possible matches, the user can click on the entry that is closest to what they intended to search, which will redirect them to a page showing information about that specific kinase/phosphosite/inhibitor. For example, these are two of the queries needed to show the information on the page for a kinase:

```
searchkin =
session.query(HumanKinases).filter(HumanKinases.Entry_name.startswith(kinase_name))
first()
```

This query is to obtain information about a specific kinase from the HumanKinases table, and in this case the query isolates the rows in which the entry on the Entry_name column starts with the kinase chosen by the user on the results page returned before. The query should return only one row (i.e. there should be only one entry for each kinase), but in case there are duplicated rows, the query is set to return the first result only.

Finally, to find by which kinases the substrates in a phosphoproteomics data file (uploaded by the user) are phosphorylated, you can read about the data analysis on point 3.3.

[redacted]

3.2.4 Uploading and downloading files

One of the requirements of the website is allowing users to upload files with phosphoproteomics data, analyze it and return the results, which can be uploaded.
The files need to be saved to run the function data_analysis on them.
A relative path to the folder in which the files uploaded are stored is defined as UPLOAD_FOLDER. The code to save the files and run the data_analysis on them is:

```
file = request.files['file']
if request.method == 'POST':
.               if file and allowed_file(file.filename):                                    .
                filename = secure_filename(file.filename)
                file.save(os.path.join(application.config['UPLOAD_FOLDER'], filename))
                data_analysis(os.path.join(application.config['UPLOAD_FOLDER'], filename),
filename)
                return redirect(url_for('uploaded_file', filename=filename))
return render_template('homepage.html', form=form)
```

The request.files object keeps files uploaded through an upload file button (which is embedded on homepage.html), and uploading files is a POST method since information is being sent from the web browser to the web server. Therefore, the 'if' instance in the code above is to set what the function should do when a file is uploaded, while otherwise a template is rendered. The function allowed_file defined prior to this code specifies which file extensions are allowed, therefore only when the file uploaded is in one of the extensions allowed, the secure_filename method is necessary to save the name of the file uploaded securely, then the file is saved and the data_analysis function run, with the path to the file as an argument. Finally, the user is redirected to the URL for uploaded_file, which shows the results of the function

In the data_analysis function, four tables are returned and saved as csv files on a folder downloads, for which the path is defined in the object DOWNLOAD_FOLDER. When the user clicks on an hyperlink to download the first table for example, the website is redirected to the URL under the following route instance:

```
def download_table1(inhibitor):
   return send_from_directory(application.config['DOWNLOAD_FOLDER'], filename='table1
   + inhibitor+'_analysis.csv' , as_attachment=True)
```

The send_from_directory method allows to send files from a directory to the user through the web browser, as long as the directory and the name of the file are specified after it.

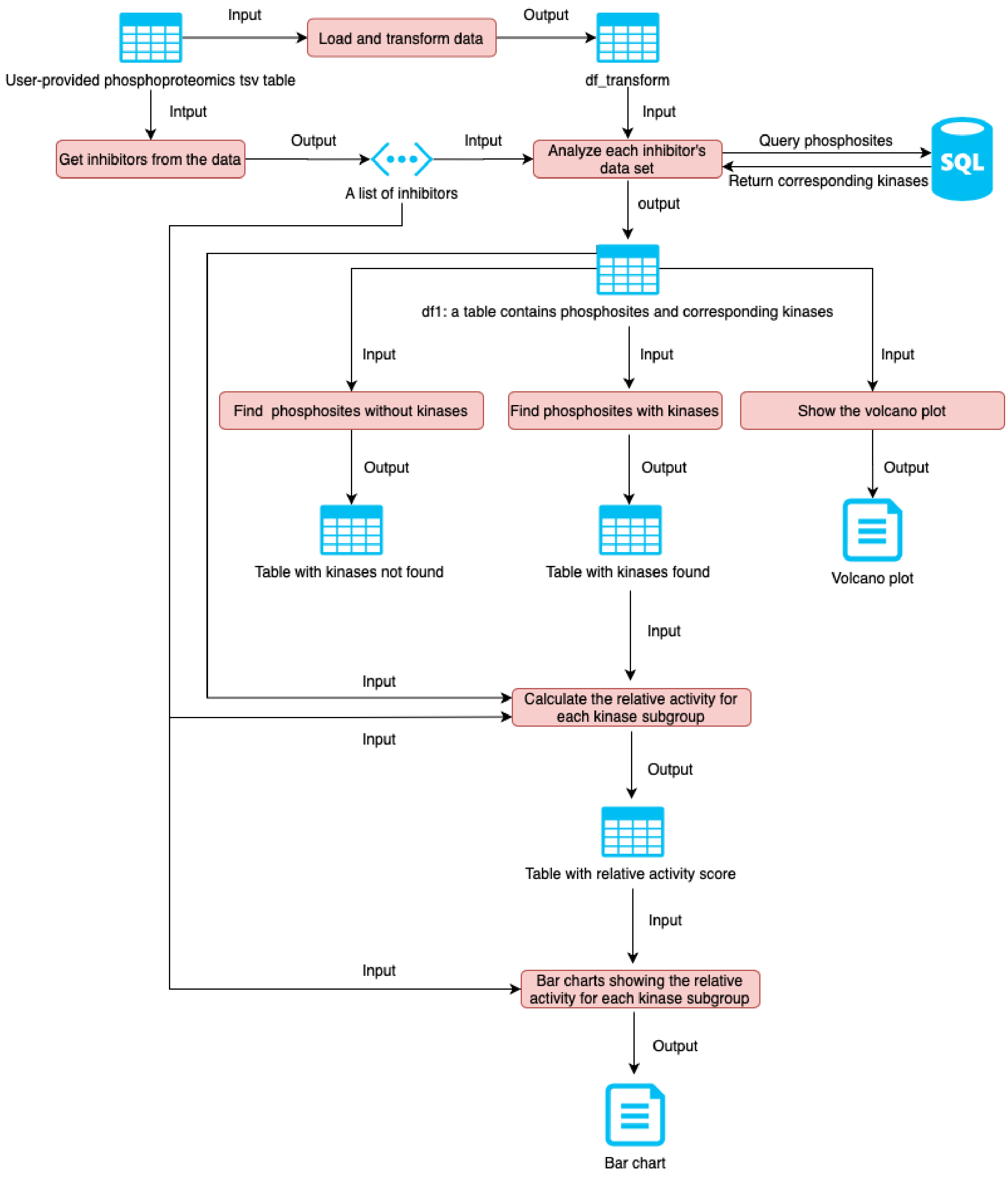
3.3 Data analysis

In order to calculate the relative activity of the kinases based on the phosphoproteomics data set, a method named KSEA (Kinase-Substrate Enrichment Analysis) is adopted in the web application. KSEA assumes that the fold changes of substrates of a specific kinase between different experimental conditions can indirectly reflect the activity of the kinase. The analysis function on the web application works as follows: first of all, by querying the database, the corresponding kinase of each substrate in the phosphoproteomics data set can be found. Using this information, the substrates are arranged into different kinase subgroups. Next, a mean log fold change of substrates in each subgroup is calculated and this mean will be divided by the mean log fold change of all substrates in the phosphoproteomics data set to yield the kinase relative activity score (RAS). Finally, the web application returns a RAS table and an interactive bar chart to the client.

The input data set for the analysis should be a tsv (tab-separated) file which contains seven columns, including the substrate (phosphosite) ID, the substrate abundance in control condition, the substrate abundance in treated condition, the fold change of abundance between the two conditions, the p value of the fold change, coefficients of variance for replicates in control condition and the coefficients of variance for replicates in treated condition (the last two columns are optional). These columns should be put in the same order as described above. Alternatively, for multi-inhibitor datasets, columns 1-2 must only be present once, columns 6-7 are optional, and columns 3-5 (or 3-7) should be repeated once for each inhibitor.

There are three main outputs that clients should focus on. Firstly, the analysis returns a volcano plot summarizing the whole phosphoproteomics data set. A bonferroni-corrected threshold is applied to the volcano plot to exhibit the substrates with significant fold changes between conditions. Secondly, the analysis returns a table containing the RAS for each kinase identified in the database. This table can be downloaded as a csv file. Thirdly, an interactive bar chart visualizing the relative activity score is returned. Clients can put the cursor on each bar to see more details of each kinase subgroup.

Additionally, the web application informs the client of the phosphosites that it was unable to locate a kinase for in the database. This ensures transparency about the basis on which the RAS were calculated. The client has the option to download three phosphosite tables: one listing every phosphosite in their original file (excluding un-phosphorylated peptides), one listing only the matched phosphosites and one listing only the unmatched phosphosites.



Input
Load and transform data
Output
User-provided phosphoproteomics tsv table
df_transform
Intput
Input
Get inhibitors from the data
Output
Intput
Analyze each inhibitor's data set
Query phosphosites
SQL
Return corresponding kinases
A list of inhibitors
output
df1: a table contains phosphosites and corresponding kinases
Input
Input
Input
Find phosphosites without kinases
Find phosphosites with kinases
Show the volcano plot
Output
Output
Output
Table with kinases not found
Table with kinases found
Volcano plot
Input
Input
Calculate the relative activity for each kinase subgroup
Input
Output
Table with relative activity score
Input
Input
Bar charts showing the relative activity for each kinase subgroup
Output
Bar chart

Figure 6. Workflow of the KSEA in JACKY

3.4 On-line deployment of the application

JACKY was deployed on Amazon Web Services Elastic Beanstalk (Free Tier). We used the Web Server environment tier and the preconfigured platform Python 3.6, running on 64bit Amazon Linux/2.9.5. The code was uploaded in a zip file as per the file structure in 3.5

3.5 Software architecture

The files will be structured in a single folder, as follows:

```
.
├── .ebextensions/
│ └── environmentvariables.config
├── downloads/
├── static/
│ └── aesthetic.css
│ └── Chromosomes1.jpg
│ └── Chromosomes12.jpg
│ └── file_structure.jpg
│ └── JACKY.jpg
│ └── RKA_table.jpg
├── templates/
│ └── datanalysis.html
│ └── homepage.html
│ └── homepageinh.html
│ └── homepagekin.html
│ └── homepagephos.html
│ └── inhibitorhits.html
│ └── inhibitorpage.html
│ └── kinasehits.html
│ └── kinasepage.html
│ └── phoshits.html
│ └── phosphositepage.html
│ └── zphoshitsgen.html
│ └── zphoshitsgenalt.html
├── uploads/
├── application.py
├── c__sqlite_final_database_v8.db
├── requirements.txt
```

4. Requirements

In order to run the software locally or on Amazon Web Services, the environment requires the following Python packages:

```
certifi==2019.11.28
Click==7.0
Flask==1.1.1
Flask-WTF==0.14.2
itsdangerous==1.1.0
Jinja2==2.11.0
MarkupSafe==1.1.1
numpy==1.18.1
pandas==1.0.0
plotly==4.5.0
python==3.6
python-dateutil==2.8.1
pytz==2019.3
retrying==1.3.3
scipy==1.4.1
six==1.14.0
SQLAlchemy==1.3.12
Werkzeug==0.16.1
```

[redacted]

wincertstore==0.2
WTForms==2.2.1

jquery 3.4.1 was also used when developing the website, but was not required in the AWS environment.

In order to run the scripts to produce the database and associated csv input files, the following packages were required:

chembl_webresource_client.new_client (no version)
csv 1.0
pandas 0.25.1
re 2.2.1
sqlalchemy 1.3.9
sqlalchemy.ext.declarative (no version)
sqlalchemy.orm (no version)
urllib.parse (version not available)
urllib.request 3.7

5. Data flow

The data follows three different pathways:

- Extraction of data from external sources and transfer to relational database using python scripts (Figure 7). This process is done once to feed the database to use in the software.

- Queries made by the user (Figure 7). The user enters search terms and, for phosphosite search, selects the search method. The queries interrogate the database for the specified request and return an output in a new window where the user can visualise the data.

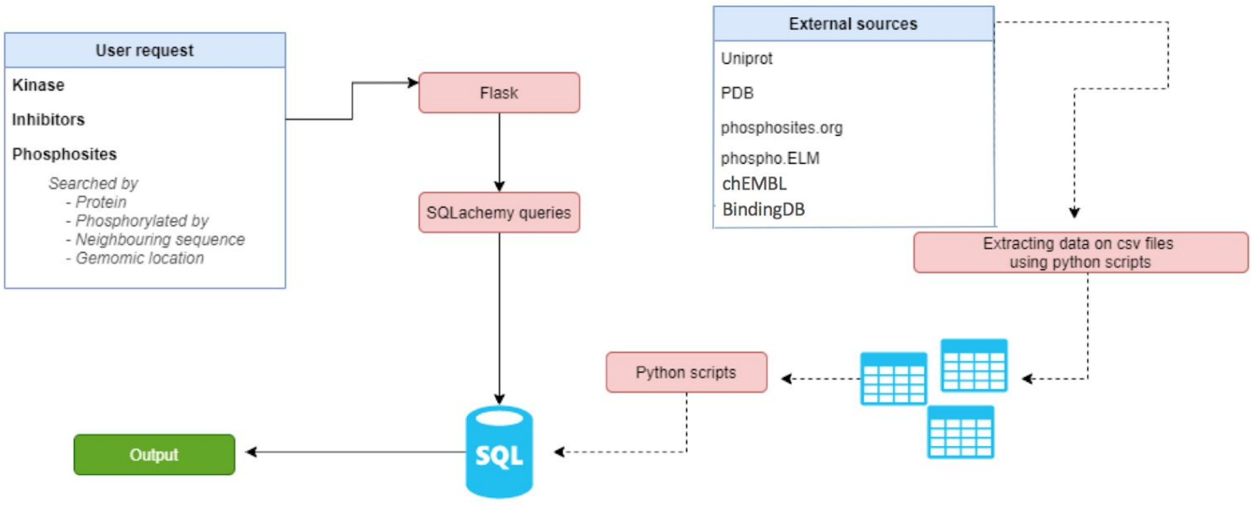


Figure 7. Overall data flow of the software

[redacted]

6. Limitations and prospective work

6.1 Limitations

The accuracy and completeness of the data in JACKY depends on a) the information collected from the original sources: UniProt, PDB, PhosphoSitePlus, phospho.ELM, BindingDB, chEMBL and UniChem, and b) the ability of our developers to make links between kinases and phosphosites, and kinases and inhibitors, unambiguously. The data in JACKY version 1 was retrieved in January and February 2020. Updates of information in these repositories will not be included in the database automatically. This should be done at least twice a year to make sure the updates on the external sources are incorporated.

6.2 Prospective work

At the moment JACKY allows the user to upload experimental results to carry out an evaluation using data from the database and download those results. In the future, a system to store those research data and share with other users could be implemented. For that, the research results would need to have been peer-reviewed and published which will grant the quality of those for other users to use them for further analysis. The owner of the data will need to agree its use.

More UniProt IDs could be identified for phospho.ELM kinases using more sophisticated text mining.

User-submitted parameters for p-value significance thresholds, and coefficient of variance cutoffs (to exclude sample means yielded from replicates with high variance) could be incorporated into the statistical analysis.

A "shopping cart" feature could be added to allow users to select and store kinases, phosphosites, inhibitors and diseases of interest in downloadable tables.

Many of the human kinases from UniProt were mapped to several PDB structures, but in this version, only one was selected, and others removed. In future versions, a section of the website designated to the kinase structure could be added and a drop down menu of all the possible structures associated with that human kinase could be added. Therefore the user can view all the protein structures and not just one. Furthermore, not all human kinases could be mapped from UniProt to PDB, and therefore some kinases were left without a structure on the website. For some kinases, however, a mouse, or other model organism, structure was available, and therefore in future releases, for those human kinases without a PDB structure, a model organism structure could be used to replace it. If more than one structure is available for the model organism kinase, then a drop down menu window could also be introduced for this.

More detailed feedback of unexpected error messages for clients could be added to the application.

The datatable plugin was successfully implemented on the Targets table on the kinase information page but not on the tables on the phosphosite and inhibitor information pages.

7. Contacts

JACKY was designed and the developed in January and February 2020 by the students of the MSc in Bioinformatics at the Queen Mary University of London:

Juan Ledesma Moreno (j.ledesmamoreno@se19.qmul.ac.uk)

Anna Ruth Dearman (a.dearman@se19.qmul.ac.uk)

Celia De Los Angeles Colomina Basanta (c.colominabasanta@se16.qmul.ac.uk)

Katie Tamara Skinner (k.skinner@se19.qmul.ac.uk)

Yutang Chen (y.chen@se19.qmul.ac.uk)

Please feel free to contact us for support, feedback or any comment that you consider could be done to improve the software. Your opinion is important for JACKY.

8. References

Anaconda | The World’s Most Popular Data Science Platform (no date). Available at: https://www.anaconda.com/ (Accessed: 13 February 2020).

AWS Elastic Beanstalk – Deploy Web Applications (no date). Available at: https://aws.amazon.com/elasticbeanstalk/ (Accessed: 13 February 2020).

Benfield, C., Robenolt, M. and Cordasco, IS., Certifi (2020), GitHub repository, https://github.com/certifi/

Berman, H. M. (2000) ‘The Protein Data Bank / Biopython’, Presentation. Oxford University Press, 28(1), pp. 235–242. doi: 10.1093/nar/28.1.235.

Bishop, S., pytz, (2020), GitHub repository, https://github.com/newvem/pytz

Chambers, J. et al. (2013) ‘UniChem: A unified chemical structure cross-referencing and identifier tracking system’, Journal of Cheminformatics, 5(1). doi: 10.1186/1758-2946-5-3.

Davies, M. et al. (2015) ‘ChEMBL web services: streamlining access to drug discovery data and utilities’, Nucleic Acids Research. Oxford University Press, 43(W1), pp. W612–W620. doi: 10.1093/nar/gkv352.

Dinkel, H. et al. (2011) ‘Phospho.ELM: a database of phosphorylation sites--update 2011’, Nucleic Acids Research. Narnia, 39(Database), pp. D261–D267. doi: 10.1093/nar/gkq1104.

Gaulton, A. et al. (2017) ‘The ChEMBL database in 2017’, Nucleic Acids Research. Oxford University Press, 45(D1), pp. D945–D954. doi: 10.1093/nar/gkw1074.

Gilson, M. K. et al. (2016) ‘BindingDB in 2015: A public database for medicinal chemistry, computational chemistry and systems pharmacology’, Nucleic Acids Research. Oxford University Press, 44(D1), pp. D1045–D1053. doi: 10.1093/nar/gkv1072.

Holder, R., McClosky, A., Dunkelberger, J., Arthur, JT., Wilson, JD., Kuang, A., Dollé, S., Dooley, R., Shanabrook, S., Nephin, D., Visser, S., Harlow, J., Chibon, P., Guémar, H., Goirand, T., Page, J., Marshall, J., Matthews, D., Taylor, M., Shalenyi, M., Herriott, J., Evers, J. and Durgin, C., retrying, (2020) GitHub repository, https://github.com/rholder/retrying

Hornbeck, P. V. et al. (2015) ‘PhosphoSitePlus, 2014: Mutations, PTMs and recalibrations’, Nucleic Acids Research. Oxford University Press, 43(D1), pp. D512–D520. doi: 10.1093/nar/gku1267.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederoc, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., Jupyter Development Team (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows. IOS Press. doi: 10.3233/978-1-61499-649-1-87

Oliphant, T.E., 2006. A guide to NumPy, Trelgol Publishing USA.

Pedro, C. et al. (2013) ‘Kinase-Substrate Enrichment Analysis Provides Insights into the Heterogeneity of signaling Pathway Activation in Leukemia Cells’, SCIENCESIGNALING, Vol 6 Issue 268 rs6

Plotly Technologies Inc. (2015) Collaborative data science Publisher: Plotly Technologies Inc. Montréal, QC. URL: https://plot.ly

PyData Development Team (2020). Powerful data structures for data analysis, time series, and statistics. Python Software Foundation. URL http://pandas.pydata.org

Python Core Team (2020). Python: A dynamic, open source programming language. Python Software Foundation. URL https://www.python.org/.

[redacted]

The Architecture of Open Source Applications (Volume 2): SQLAlchemy (no date). Available at: http://aosabook.org/en/sqlalchemy.html (Accessed: 13 February 2020).

Virtanen, P. et al. (2020) ‘SciPy 1.0: fundamental algorithms for scientific computing in Python.’, Nature methods. Nature Publishing Group, pp. 1–12. doi: 10.1038/s41592-019-0686-2.

Van Der Walt, S., Colbert, S. C. and Varoquaux, G. (2011) ‘The NumPy array: A structure for efficient numerical computation’, Computing in Science and Engineering, 13(2), pp. 22–30. doi: 10.1109/MCSE.2011.37.

The UniProt Consortium (2019) ‘UniProt: a worldwide hub of protein knowledge’, *Nucleic Acids Research*, 47(1), pp. 506-515

Various authors, chembl_webresource_client (2020), GitHub repository, https://github.com/chembl/chembl_webresource_client

Various authors, click, (2020), GitHub repository, https://github.com/pallets/click

Various authors, dateutil, (2020), GitHub repository, https://github.com/dateutil/dateutil

Various authors, flask, (2020), GitHub repository, https://github.com/pallets/flask

Various authors, flask-wtf, (2020), GitHub repository, https://github.com/lepture/flask-wtf

Various authors, flask-sqlalchemy, (2020), GitHub repository, https://github.com/pallets/flask-sqlalchemy

Various authors, itsdangerous, (2020), GitHub repository, https://github.com/pallets/itsdangerous

Various authors, jinja, (2020), GitHub repository, https://github.com/pallets/jinja

Various authors, jquery, (2020), GitHub repository, https://github.com/jquery/jquery

Various authors, markupsafe, (2020), GitHub repository, https://github.com/pallets/markupsafe

Various authors, re (2020), GitHub repository, https://github.com/python/cpython/blob/3.8/Lib/re.py

Various authors, six, (2020), GitHub repository, https://github.com/benjaminp/six

Various authors, urllib.parse (2020), GitHub repository, https://github.com/python/cpython/blob/3.8/Lib/urllib/parse.py

Various authors, urllib.request (2020), GitHub repository, https://github.com/python/cpython/blob/master/Lib/urllib/request.py

Various authors, werkzeug, (2020), GitHub repository, https://github.com/pallets/werkzeug

Various authors, wincertstore, (2020), GitHub repository, https://github.com/mindw/wincertstore

Various authors, wtforms, (2020), GitHub repository, https://github.com/wtforms/wtforms

Wirbel, J., Cutillas, P. and Saez-Rodriguez, J. (2018) ‘Phosphoproteomics-based profiling of kinase activities in cancer cells’, in Methods in Molecular Biology. Humana Press Inc., pp. 103–132. doi: 10.1007/978-1-4939-7493-1_6