**Kellerman Kinase Database Documentation**

Contents

[Overview 2](#_Toc787120)

[Database Structure 2](#_Toc787121)

[Database schema 2](#_Toc787122)

[Software Structure 4](#_Toc787123)

[Files in the Kellerman repository 4](#_Toc787124)

[Run.py 4](#_Toc787125)

[App.py 4](#_Toc787126)

[Config.py 5](#_Toc787127)

[Procfile 5](#_Toc787128)

[Requirements.txt 5](#_Toc787129)

[.gitignore 6](#_Toc787130)

[Folders in the Kellerman Repository 6](#_Toc787131)

[Common 6](#_Toc787132)

[Models 6](#_Toc787133)

[Downloads 7](#_Toc787134)

[Templates 7](#_Toc787135)

[Static 7](#_Toc787136)

[Uploads 8](#_Toc787137)

[Limitations and Next Steps 8](#_Toc787138)

[References 8](#_Toc787139)

1. overall design philosophy
2. explain how the software is structured and how it works
3. explain the technologies that were used to develop the software

# Overview

The Kellerman Kinase application is an interactive searchable database. It provides the user with background information about protein kinases such as their gene name, the families they belong to, the cell location and their target phosphorylation sites. As well as target phosphorylation sites its also provides the user with the genomic location and neighbouring sequences of their respective phosphosites. Inhibitor information for each kinase is retrieved and this database allows you to search known inhibitors on the inhibitor page, displaying their name, chemical structure and respective kinase they are known to inhibit. Furthermore, this database contains an analysis section allowing the user to upload their own phosphoproteomics data. The analysis section calculates and graphically presents the relative active of each kinase in the sample data.

The protein kinase and phosphosite data was collected through UniProt API and inhibitor data was collected from the MRC Interational Center of Kinase Profiling. The collected data was populated into a PostgreSQL database in PgAdmin4. The application itself was built through Flask in Python 2.7 and deployed through the GitHub repository in Heroku.

# Database Structure

The data was collected through the URL API ipython notebook and saved as CSV files. The database was built in PgAdmin4 which is a requirement of Postgresql. Database schema is found in the code below and was built in PgAdmin4.

## Database schema

CREATE TABLE public."Kinase\_table"

(

"KINASE\_NAME" character varying(255) COLLATE pg\_catalog."default" NOT NULL,

"ACCESSION\_ID" character varying(255) COLLATE pg\_catalog."default" NOT NULL,

"CAP\_KINASE\_NAME" character varying(255) COLLATE pg\_catalog."default",

"UNIPROT\_ID" character varying(255) COLLATE pg\_catalog."default",

"GROUP" character varying(255) COLLATE pg\_catalog."default",

"FAMILY" character varying(225) COLLATE pg\_catalog."default",

"SUBFAMILY" character varying(255) COLLATE pg\_catalog."default",

"PROTEIN\_SEQUENCE" character varying(100000) COLLATE pg\_catalog."default",

"KINASE\_DOMAIN" character varying(100000) COLLATE pg\_catalog."default",

"ENTREZ\_ID" character varying(255) COLLATE pg\_catalog."default",

"CHROMOSOME" character varying(255) COLLATE pg\_catalog."default",

"CHROMOSOME\_LOC" character varying(1000) COLLATE pg\_catalog."default",

"SUBCELLULAR\_LOC" character varying(10000) COLLATE pg\_catalog."default",

"SYNONYMS" character varying(10000) COLLATE pg\_catalog."default",

CONSTRAINT "AccessionID" PRIMARY KEY ("ACCESSION\_ID")

CREATE TABLE public."Inhibitor\_table"

(

"INHIBITOR\_NAME" character varying(255) COLLATE pg\_catalog."default" NOT NULL,

"CHEMICAL\_STRUCTURE" character varying(255) COLLATE pg\_catalog."default",

"MOLECULAR\_WEIGHT" character varying(255) COLLATE pg\_catalog."default",

"CNUMBER" character varying(255) COLLATE pg\_catalog."default" NOT NULL,

"ACTION" character varying(255) COLLATE pg\_catalog."default",

"INCHL\_KEY" character varying(255) COLLATE pg\_catalog."default",

"PUBCHEM\_CID" character varying(20) COLLATE pg\_catalog."default",

"SMILES" character varying(255) COLLATE pg\_catalog."default",

inh\_lower character varying(255) COLLATE pg\_catalog."default",

CONSTRAINT "CNumber" PRIMARY KEY ("CNUMBER")

)

CREATE TABLE public."KiInh\_relation\_table"

(

"KINASE\_NAME" character varying(255) COLLATE pg\_catalog."default",

kin\_lower character varying(255) COLLATE pg\_catalog."default",

"ACCESSION\_ID" character varying(255) COLLATE pg\_catalog."default",

"INHIBITOR\_NAME" character varying(255) COLLATE pg\_catalog."default",

"CNUMBER" character(255) COLLATE pg\_catalog."default" NOT NULL,

"SERIES\_NUM" character(255) COLLATE pg\_catalog."default" NOT NULL,

CONSTRAINT "KiInh\_relation\_table\_pkey" PRIMARY KEY ("SERIES\_NUM"),

CONSTRAINT "ACCESSION\_ID" FOREIGN KEY ("ACCESSION\_ID")

REFERENCES public."Kinase\_table" ("ACCESSION\_ID") MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT "CNUMBER" FOREIGN KEY ("CNUMBER")

REFERENCES public."Inhibitor\_table" ("CNUMBER") MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

CREATE TABLE public."Phosphosite\_table"

(

"SERIES\_NUM" character varying(255) COLLATE pg\_catalog."default" NOT NULL,

"ENSEMBL\_ID" character varying(255) COLLATE pg\_catalog."default",

"GENE\_NAME" character varying(255) COLLATE pg\_catalog."default",

"GENE\_START(BP)" character varying(255) COLLATE pg\_catalog."default",

"GENE\_END(BP)" character varying(255) COLLATE pg\_catalog."default",

"CHROMOSOMAL\_LOC" character varying(255) COLLATE pg\_catalog."default",

"PHOSPHORYLATION\_LOC" character varying(255) COLLATE pg\_catalog."default",

"KINASE\_NAME" character varying(255) COLLATE pg\_catalog."default",

kin\_lower character varying(255) COLLATE pg\_catalog."default",

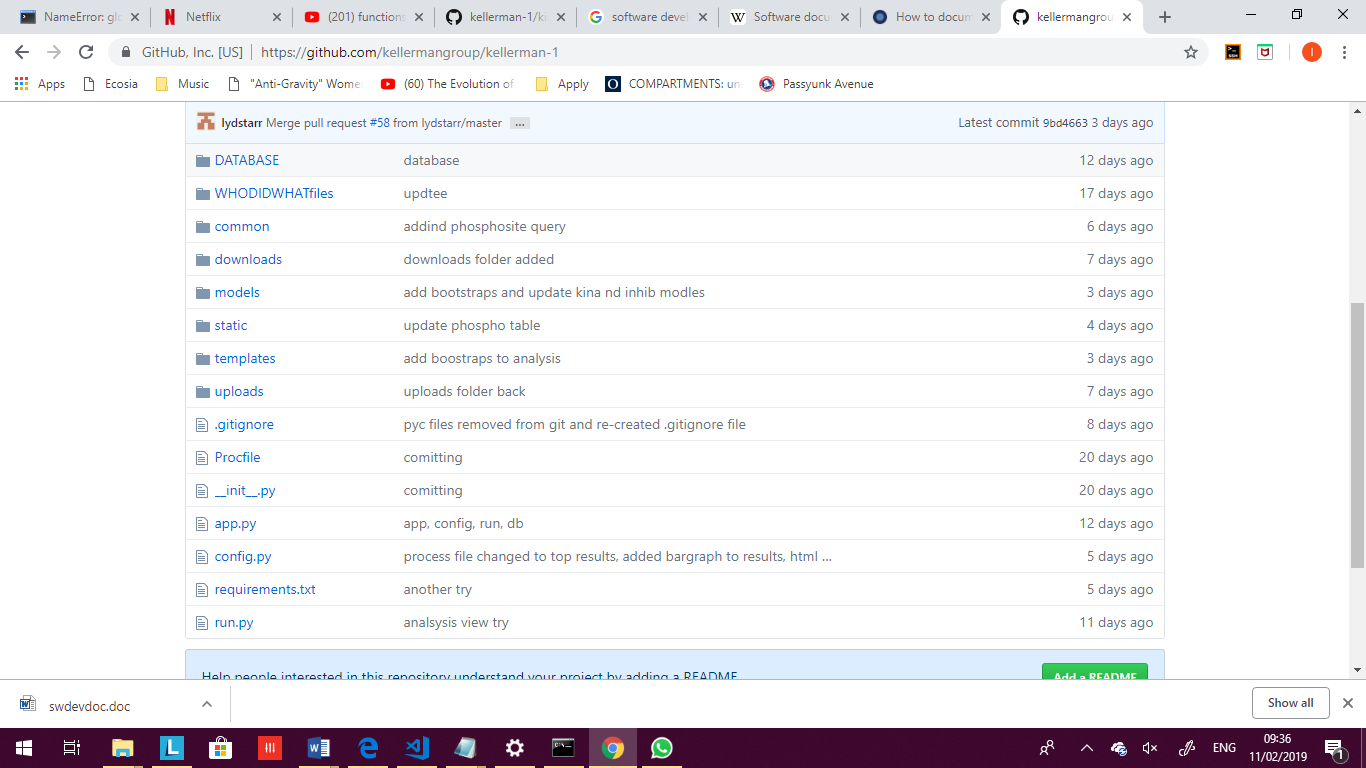
CONSTRAINT "SERIES\_NUM" PRIMARY KEY ("SERIES\_NUM")

)

The tables were populated with the import csv function in PgAdmin4 and were quried in the application software.

# Software Structure

The software was structured through organised folders and files stored in a GitHub repository. Figure 1 shows a screenshot of the software structure. In each folder, there is a respective \_\_init\_\_.py file to mark directories as python packages when the app is running and should remain empty.



**Figure 1.** Structure of the software files and folder in GitHub

## Files in the Kellerman repository

### Run.py

* The file runs the app either locally or deployed. It imports the operating system (os) to set the environment to run through a port and it also imports the app to apply debug mode.

""" This File is used to run the app deployed or locally"""

import os

from app import APP

port = int(os.environ.get('PORT', 33506))

# run deployed

APP.run(debug=APP.config['DEBUG'], use\_reloader=APP.config['RELOADER'], port=port, host='0.0.0.0')

### App.py

* The app.py file imports the Flask module and render template. The application is created by applying a variable to the Flask module.

# import Flask and render template

from flask import Flask, render\_template

# create a flask application

APP = Flask(\_\_name\_\_)

APP.config.from\_object('config')

* The app.py file contains the application home route index in which renders the home index template. Moreover, it imports Flask blueprints which connects all the route pages through the models folder to the respective views.py files.

# import blueprints from models folder

from models.kinase.views import kinase\_blueprint

from models.inhibitor.views import inhibitor\_blueprint

from models.analysis.views import analysis\_blueprint

# define action of top level routes

@APP.route("/")

def index():

""" Description: Home Route that renders the index template """

return render\_template('index.html')

# register other route blueprints

APP.register\_blueprint(kinase\_blueprint, url\_prefix="/kinase")

APP.register\_blueprint(inhibitor\_blueprint, url\_prefix="/inhibitor")

APP.register\_blueprint(analysis\_blueprint, url\_prefix="/analysis")

### Config.py

* The config file sets the configurations needed for the app.

""" Define All essential configurations we will need for the app itself"""

import os

DEBUG = True # Turns on debugging features in Flask

RELOADER = True

UPLOAD\_FOLDER = os.path.dirname(os.path.abspath(\_\_file\_\_)) + '/uploads'

DOWNLOAD\_FOLDER = os.path.dirname(os.path.abspath(\_\_file\_\_)) + '/downloads'

STATIC\_FOLDER = os.path.dirname(os.path.abspath(\_\_file\_\_)) + '/static'

### Procfile

* Procfile is linked to Heroku and is required to specify the commands to execute on the app start up.

web: python run.py

### Requirements.txt

Flask

psycopg2

Flask-SQLAlchemy

### .gitignore

* .gitignore file is required to tell git which commits to ignore and is not necessarily required for the application itself.

## Folders in the Kellerman Repository

### Common

* In the common folder is the database.py file that connects to the Postgresql database server through pyscopg2. There are 3 functions in this folder that run queries for retrieving kinase, inhibitor, and phosphosite data, which are imported into their respective routes.

import psycopg2

# Example of query function

def Query(query):

# Define our connection string

conn\_string = ("host='ec2-54-75-245-94.eu-west-1.compute.amazonaws.com' dbname='d71uh4v1fd2hq' user='tdsneouerzmxkj' password='92a500cb091fe70168b32c66fa6a3d6c376d467d57fb9b663eb5d13446ecb2e6'")

try:

# get a connection, if a connect cannot be made an exception will be raised here

conn = psycopg2.connect(conn\_string)

# conn.cursor will return a cursor object, you can use this cursor to perform queries

cur = conn.cursor()

cur.execute(query)

query = cur.fetchall() # fetch query

conn.close() # close connection

return query

except:

return 'An error occured while executing SQL query'

### Models

* The models folder contains the views.py files for each route. It consists of 3 further folders for each route; analysis, kinase, and inhibitors. The blueprints for each route are in the respective views.py folder.
  + The kinase and inhibitor index page contain a search request form for the user to input their search. The input is stored as the nameFilter variable which is parsed into the database queries. Each index route renders the respective index template. The route is redirected to the results page when the search has been submitted. The code for the kinase route index is found below.

# import requirements for the model to run

from flask import Flask, Blueprint, render\_template, request, url\_for, redirect

# import database from common database

import common.database as db

# name kinase blueprint

kinase\_blueprint = Blueprint('kinase', \_\_name\_\_)

# make search box using request form for the index page

@kinase\_blueprint.route('/', methods=['GET', 'POST'])

def index():

if request.method == 'POST':

if request.form['name']:

# make user input a variable named nameFilter

nameFilter = request.form['name']

# turn kinase name into uppercase

nameFilter = nameFilter.upper()

# redirect to results page

return redirect(url\_for('kinase.results', nameFilter=str(nameFilter)))

return render\_template('kinase/index.html')

* + The results section for the kinase and inhibitor routes call the queries from the common database file and gives an output variable as data which is entered into the respective results html page.
* The analysis models folder contains the route index views file and a process file. The index route contains an upload form that accepts a tsv file.

### Downloads

* The downloads folder

### Templates

* The templates folder holds all the html code for the app. The menu.html file is the initial page that sets the html paraments for the entire app. It is styled with Bootstrap basic template enabling clarity for the app across different platforms. It sets the settings for the top navigation bar to direct to the different routes.
* Each html file is an extension of the menu page. The index file is the homepage for the home route, and there is a respective index page for the route index’s.

### Static

* This folder contains the style of the application in CSS. The style.css file is an addition to the Bootstraps file and applies various settings for each component in the HTML files. These file is in the static folder as they do not to be changed.

### Uploads

# Limitations and Next Steps

Using Heroku as a deployment platform for the database has its limitations. A maximum of 10 000 rows was reached as we populated the database with more information.

# References