# Concept Library Installation Guide

This document outlines the instructions to get the concept library working on your local machine.

# Prerequisites

Ensure you have the following installed:

Python 3.9

Pip

PostgreSQL version 9.4

PGAdmin3

Eclipse

Git

# Get Concept-library code from git

Open command window as administrator

Navigate to your folder where you store all of your code. For example:

C:/dev/

Then type the following command:

git clone <https://github.com/SwanseaUniversityMedical/concept-library.git>

Change directory to the newly created folder in this case concept-library.

To begin with, you will want to work on the master branch so you will need to move to the master branch by typing the following in the command window:

git checkout master

## Install virtualenv and virtualenvwrapper

This will provide a dedicated environment for each project you create. It is considered best practice and will save time when you’re ready to deploy your project.

Open your command window and type:

pip install virtualenvwrapper-win

Now change directory to where you have downloaded the project e.g. C:\Dev\concept-library

To create a virtualenv to install all of your packages for your project type:

mkvirtualenv cllproject

Note:

To work on a virtual environment to install packages type:

workon <virtualenv name>

To cancel working within a virtualenv type:

deactivate <virtualenv name>

## Install ldap functionality

For windows machines I had to install Microsoft Visual C++ compiler

For python 3.9

pip install --upgrade setuptools

https://visualstudio.microsoft.com/visual-cpp-build-tools/

https://www.lfd.uci.edu/~gohlke/pythonlibs/#python-ldap

for python 2.7

<https://www.microsoft.com/en-us/download/details.aspx?id=44266>

python\_ldap-3.3.1-cp39-cp39-win\_amd64.whl is found in the folder ‘requirements’

Within your virtualenv run the following command (change directory to ‘requirements’):

pip install python\_ldap-3.3.1-cp39-cp39-win\_amd64.whl

Then run:

pip install django-auth-ldap

For reference see:

<https://django-auth-ldap.readthedocs.io/en/1.2.x/install.html>

If we are to use ldap over ssl then we’ll need to follow this example:

<https://support.microsoft.com/en-us/help/938703/how-to-troubleshoot-ldap-over-ssl-connection-problems>

## Installation of Required Packages

To install the required packages there is a txt file containing all the required packages, this makes it easy to install correct versions of each package, please enter the following into the command line:

pip install –r requirements/local.txt

This is the equivalent of installing:

> pip install Django==1.11.3

> pip install django-mathfilters==0.4.0

> pip install django-simple-history==1.9.0

> pip install djangorestframework==3.6.3

> pip install pathlib==1.0.1

> pip install psycopg2==2.7.1

> pip install pytz==2017.2

Etc ...

Note:

Within the requirements folder there are more .txt for staging and production in case they ever did differentiate for each environment.

# Concept Library Data Base Installation Guide

This document outlines the instructions to get the concept library data base working on your local machine.

The backup DB contains the coding systems and some users to try different access roles, in addition to demo data (concepts, phenotypes, etc.)

# Prerequisites

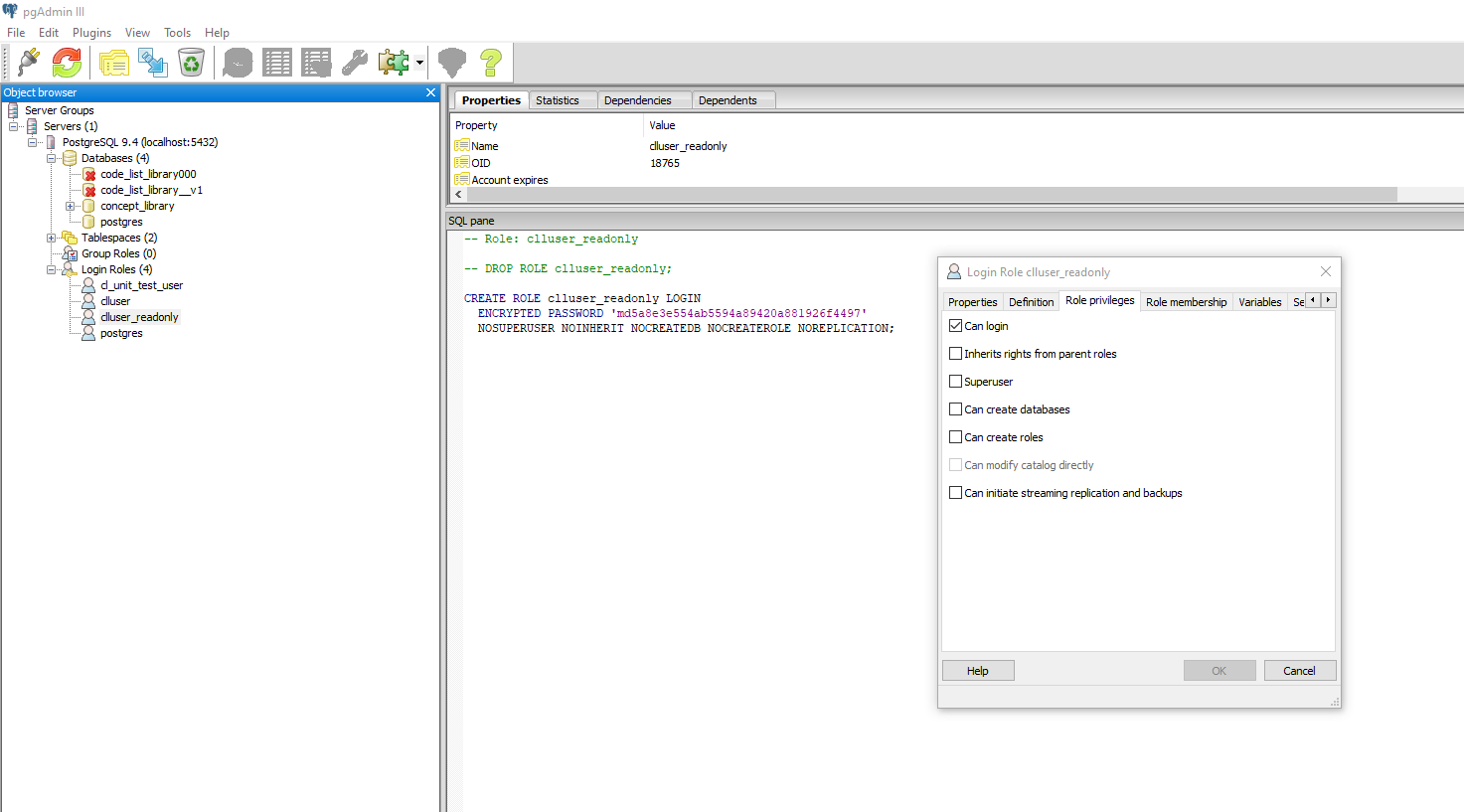
Ensure you have the following installed:

**PostgreSQL 9.4** (needs to be v9.4 to be the same as the production server)

**PGAdmin 3** (higher PGAdmin sometimes give error messages with PostgreSQL 9.4)

# Database setup

Using PGAdmin3

1. Create a role called **clluser**
2. Create a database called **concept\_library** (owner= postgres)
3. Create a read-only role called **clluser\_readonly**
   1. The read-only role must have (NOSUPERUSER NOINHERIT NOCREATEDB NOCREATEROLE NOREPLICATION)  
      
4. Create a testing role called **cl\_unit\_test\_user** (must have createdatabase)
5. Restore the DB backup using PGAdmin interfaced
6. Grant the read-only user privileges

When running the application it complains that you have unapplied migrations; your app may not work properly until they are applied. So in the command window navigate to where your manage.py file is located and type:

python manage.py makemigrations

python manage.py migrate

Please see Appendix A for more information about using data migrations within Django.

It may also be a good idea at this point to reseed the database if it is a new install. There is a script within \CodeListLibrary\_root\Scripts\Reseed database.txt

To run the application from the command window you need to change directory to where the manage.py file exists and then type:

python manage.py runserver

Press Ctrl + break to stop server

# Administration Area

There are no users in your database. So we need to create a superuser in order to access the administration site to manage other users.

Open the command line and execute:

python manage.py createsuperuser

Fill in the desired username, email and password

When the development server is running, you can access the admin section by going to the following url:

<http://127.0.0.1:8000/admin/>

-- grant permissions to the clluser\_readonly user

GRANT CONNECT ON DATABASE concept\_library TO clluser\_readonly;

-- This assumes you're actually connected to concept\_library..

GRANT USAGE ON SCHEMA public TO clluser\_readonly;

-- grant select on all tables

GRANT SELECT ON ALL TABLES IN SCHEMA public TO clluser\_readonly;

-- to make this grant applied to future created tables

ALTER DEFAULT PRIVILEGES IN SCHEMA public

GRANT SELECT ON TABLES TO clluser\_readonly;

-- grant all on Django tables: since Django modifies these tables

GRANT ALL ON public.auth\_group TO clluser\_readonly;

GRANT ALL ON public.auth\_group\_permissions TO clluser\_readonly;

GRANT ALL ON public.auth\_permission TO clluser\_readonly;

GRANT ALL ON public.auth\_user TO clluser\_readonly;

GRANT ALL ON public.auth\_user\_groups TO clluser\_readonly;

GRANT ALL ON public.auth\_user\_user\_permissions TO clluser\_readonly;

GRANT ALL ON public.django\_admin\_log TO clluser\_readonly;

GRANT ALL ON public.django\_content\_type TO clluser\_readonly;

GRANT ALL ON public.django\_migrations TO clluser\_readonly;

GRANT ALL ON public.django\_session TO clluser\_readonly;

* The applications has 2 different types of users
  1. **~~Superuser~~** ~~(given only for sys admin)  
     can access everything without placing permissions. The DB backup contains a super user to be used (username:~~ **~~cl.superuser~~** ~~with password:~~ **~~Password#567~~**~~)~~AS a developer, you have a superuser account on Demo which you should use it here
  2. **Normal user (given to all users)**  
     can access what he/she creates or given access to. The DB backup contains a normal user to be used (username: **cl.normaluser** with password: **Password#567**)
* Generally, when developing, it is better to use the normal user more often to see how normal users interact with the system, as using superuser sometimes can lead to issues because it does not require permissions.
* The read-only user/read-only version must be maintained in all cases. It is required by policy that the version available inside the SAIL safe haven to be read-only (no create/update/delete/upload, or anything that can be used to copy data out of SAIL gateway).
* The application read the setting variables from env vars,

(There is a way to have these vars inside .ini file, if preferred, but the deployment will still via docker env vars)  
you need to create the following env vars

(Replace xxxxxxxxxxx with the relevant password you created)

SECRET\_KEY=amfHsyJHRRr00bd75bDDGFFf9GksfoR08jbd9876gdG5ffgEEW (just an example here)

DEBUG=True

DB\_NAME=concept\_library

DB\_USER=clluser

DB\_PASSWORD=xxxxxxxxxxx

DB\_HOST=localhost

# change this to True to enable read-only version

CLL\_READ\_ONLY=False

ENABLE\_PUBLISH=True

BROWSABLEAPI=False

SHOWADMIN=True

ALLOWED\_HOSTS=localhost, 127.0.0.1

DB\_USER\_READ\_ONLY=clluser\_readonly

DB\_PASSWORD\_READ\_ONLY=xxxxxxxxxxx

UNIT\_TEST\_DB\_NAME=cl\_test\_database

UNIT\_TEST\_DB\_USER=cl\_unit\_test\_user

UNIT\_TEST\_DB\_PASSWORD=xxxxxxxxxxx

UNIT\_TEST\_DB\_HOST=localhost

IS\_INSIDE\_GATEWAY=False

IS\_DEVELOPMENT\_PC=True

IS\_DEMO=False

# disable LDAP auth.

ENABLE\_LDAP\_AUTH=False

AUTH\_LDAP\_BIND\_PASSWORD=00000

AUTH\_LDAP\_SERVER\_URI=00000

AUTH\_LDAP\_BIND\_DN=00000

AUTH\_LDAP\_USER\_SEARCH=00000

# Set up the basic group parameters

AUTH\_LDAP\_GROUP\_SEARCH=00000

# Simple group restrictions

AUTH\_LDAP\_REQUIRE\_GROUP=00000

# Email settings

EMAIL\_HOST=00000

EMAIL\_PORT=00000

GOOGLE\_RECAPTCHA\_SECRET\_KEY= xxxxxxxxxxx

EMAIL\_HOST\_PASSWORD= xxxxxxxxxxx

EMAIL\_HOST\_USER=00000

# Get it working within Eclipse

File -> Open projects from file system

Browse to code e.g. C:\Dev\CodeListLibrary\_root\CodeListLibrary\_project\

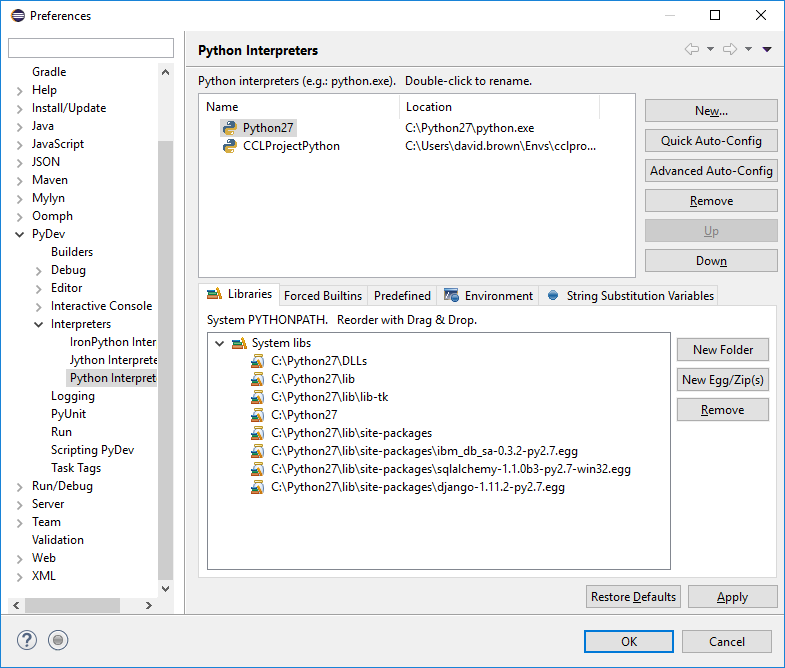
## Point your python interpreter to your VirtualEnv python Intepreter

Presuming you have created a virtualenv and installed all of your packages then you need to point your python interpreter to the virtualenv. So within Eclipse goto:

Window -> Preferences

Select

PyDev -> Intepreters -> Python Interpreter



Click New

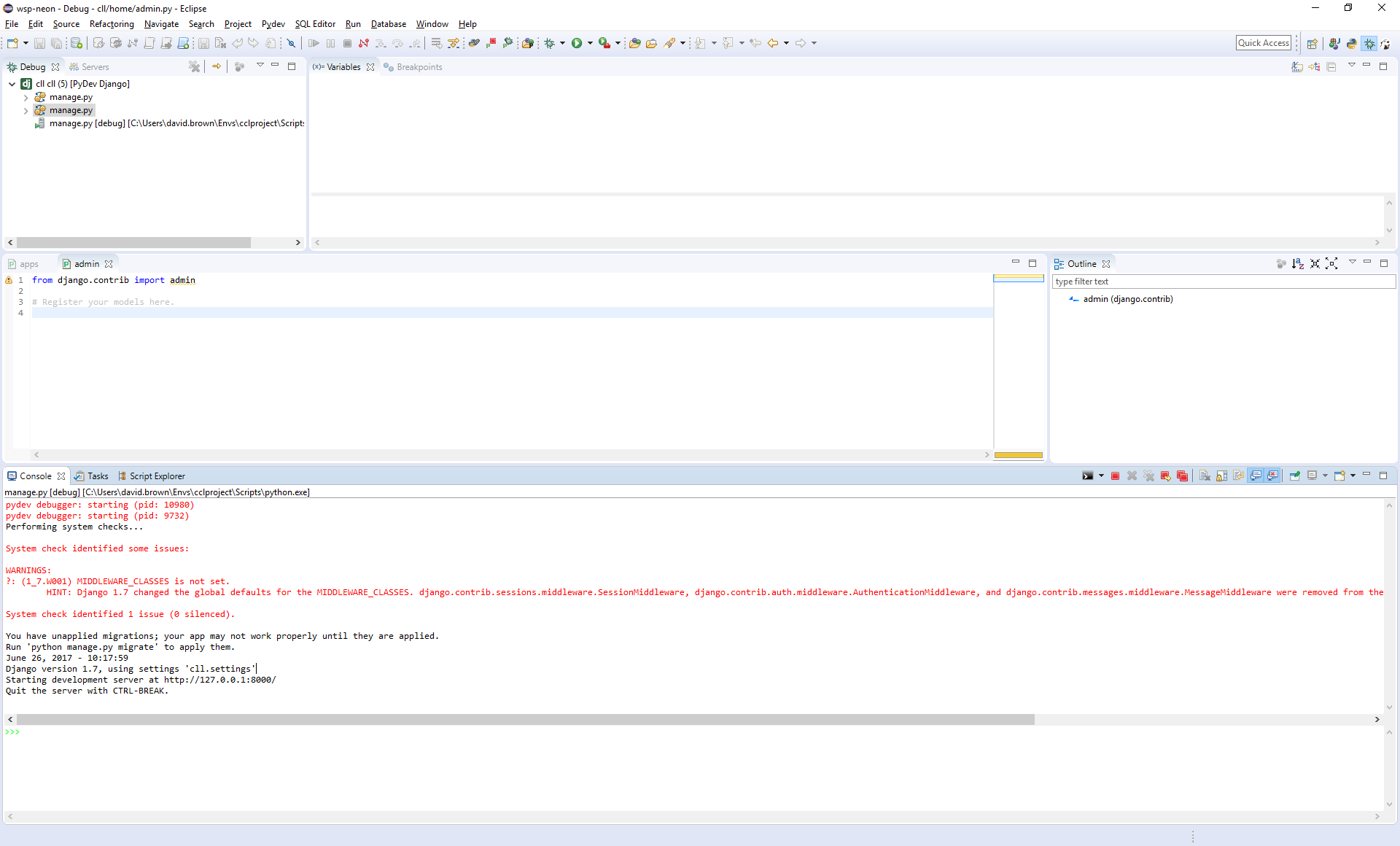
Enter Interpreter name e.g. CCLProjectPython

And browse to the python executable e.g. C:\Users\<user>\Envs\cclproject\Scripts\python.exe and then click Ok.

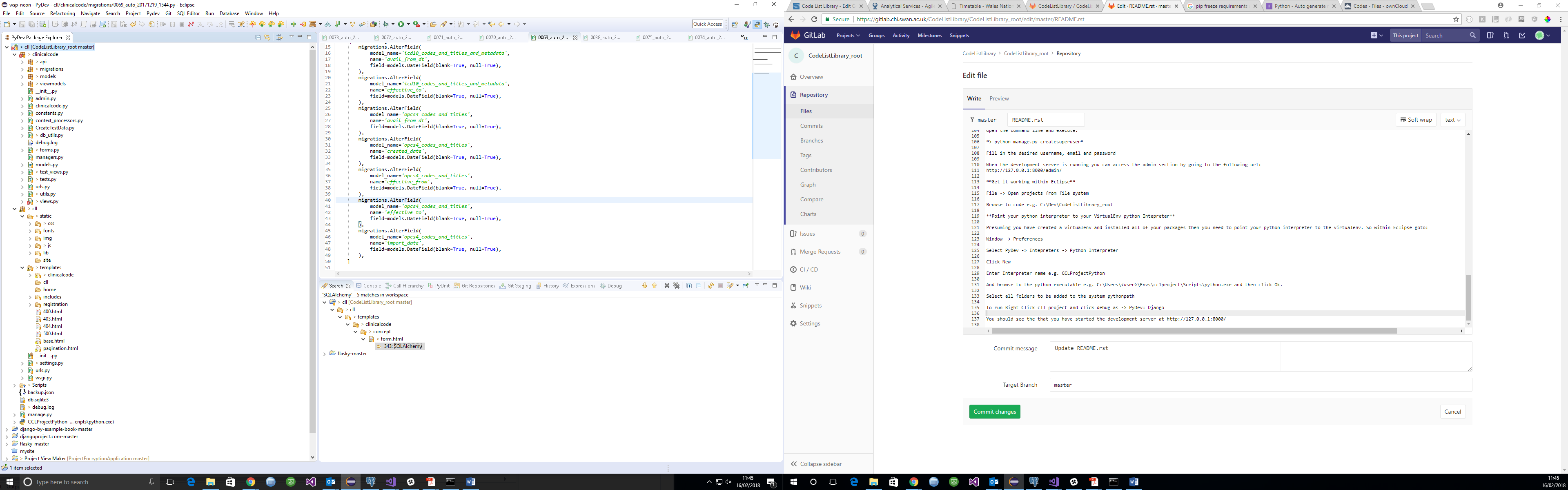
Select all folders to be added to the system pythonpath

To run right click cll project and click debug as -> PyDev: Django

You should see the that you have started the development server at <http://127.0.0.1:8000/>



Your project should look something like this in the PyDev Package Explorer:



## Extract data from SAIL Gateway

The location of the source tables are outlined below:

|  |  |
| --- | --- |
| Name | Location |
| Read codes v2 | PR\_SAIL -> SAILUKHDV.READ\_CD\_CV2\_SCD |
| Read codes v3 | PR\_SAIL -> SAILUKHDV.READ\_CD\_CV3\_TERMS\_SCD |
| OPCS4 | PR\_SAIL -> SAILUKHDV.OPCS4\_CODES\_AND\_TITLES |
| ICD10 | PR\_SAIL -> SAILUKHDV.ICD10\_CODES\_AND\_TITLES\_AND\_METADATA |

Within SAIL gateway open eclipse and select dbeaver. Connect to the PR\_SAIL database and navigate to the SAILUKHDV -> VIEWS then select the views from the table above. Right click the desired View and select Export Data -> Export to CSV file(s) then accept the defaults to save the file.

Once this is done follow the procedure to request the files out of the gateway.

**NB.** Previous developer has already done this and has exported the data to a postgresql database.

## Importing coding systems into the code list library

The codes are located at the following location:

<https://cumulus.hiru.swan.ac.uk/remote.php/webdav/SAIL_Analyst_Team/Projects/0570%20-%20Code%20List%20Library/Codes/>

Download these codes to a suitable location e.g.

C:\Users\<username>\Downloads\

Open the following files and run the queries in PGAdmin for the code\_list\_library database:

Note: You must change the location of the csv.

C:\Dev\CodeListLibrary\_root\Scripts\import ICD10 codes.sql

C:\Dev\CodeListLibrary\_root\Scripts\import OPCS4 Codes and Titles.sql

C:\Dev\CodeListLibrary\_root\Scripts\ import Read cd cv2 scd.sql

C:\Dev\CodeListLibrary\_root\Scripts\import Read cd cv3 scd.sql

# Appendix A – Data Migrations

## Keeping the database up to date

For reference:

<https://docs.djangoproject.com/en/1.11/topics/migrations/>

To propagate model changes into your database schema then you can run Django migration commands. When you make changes to your model you need to update the database so that your database matches your model. So we first need to generate the sql migrations.

In the command window navigate to where your manage.py file is located and type:

python manage.py makemigrations

Once you have your new migration files, apply them to the database by running:

python manage.py migrate

There are two other commands that might prove useful. To display the sql statement for a migration then use:

python manage.py sqlmigrate

To display a list of project migrations and their status then use:

python manage.py showmigrations

## Create Custom migration

Open the command window and navigate to the folder where manage.py is located.

Ensure you are using your working environment, if not then type:

workon cclproject

Within the command window type:

Python manage.py makemigrations clinicalcode –empty

This creates an empty migration file under:

Cll\clinicalcode\migrations\

The following file is a good example of a custom migration:

0062\_auto\_20171205\_1430.py

# Appendix B

# Create a new git application from scratch

Here are the instructions if this is a new project that is not under source control with git.

1. In GitLab ensure group and project is set up to add your code.
2. In the command window cd to where your project has been created
3. To initialize and create a local git repository type:

git init

1. Attach to the remote project on GitLab:

git remote add origin <https://gitlab.chi.swan.ac.uk/CodeListLibrary/CodeListLibrary_root.git>

1. Add all files ready for a commit

git add .

1. Commit to local repository

Git commit –m “Initial commit”

1. Push master branch to remote server:

Git push –u origin master

You know have your project under source control with git

# Appendix C – Resources

## PDFs

<http://gsl.mit.edu/media/programs/south-africa-summer-2015/materials/djangobook.pdf>

<https://doc.lagout.org/programmation/Django/Practical%20Django%20Projects%20%5BBennett%202008-06-25%5D.pdf>

<http://index-of.co.uk/Python/Packt.Publishing.Learning.Website.Development.with.Django.pdf>

## General

<https://ultimatedjango.com/learn-django/chapters/>

## Admin

<https://djangobook.com/django-admin-site/>

<https://djangobook.com/customizing-change-lists-forms/>

## Django examples

<https://github.com/mdn/django-locallibrary-tutorial/blob/master/catalog/templates/catalog/book_renew_librarian.html>

<https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Authentication>

<https://github.com/pinax/pinax-blog/blob/master/pinax/blog/forms.py>

## Forms

<http://www.tangowithdjango.com/book/chapters/forms.html>

<https://docs.djangoproject.com/en/1.11/topics/forms/modelforms/>

<https://docs.djangoproject.com/en/1.11/topics/forms/>

## CRUD

<http://www.pythondiary.com/tutorials/simple-crud-app-django.html>

<http://www.pythondiary.com/tutorials/permission-based-crud-app-django.html>

<https://simpleisbetterthancomplex.com/tutorial/2016/11/15/how-to-implement-a-crud-using-ajax-and-json.html#create-book>

<https://rayed.com/wordpress/?p=1266>

## Class Based Views

<https://micropyramid.com/blog/django-migrating-from-function-based-views-to-class-based-views/>

<https://www.codementor.io/jamesezechukwu/working-with-class-based-views-in-django-5zkjnrvwc>

<https://robots.thoughtbot.com/class-based-generic-views-in-django>

## Generic Views

<https://djangobook.com/generic-view-reference/>

## Sessions

<https://django.readthedocs.io/en/1.4.X/topics/http/sessions.html>

<https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Sessions>

## Pagination

<http://schinckel.net/2014/08/17/leveraging-html-and-django-forms%3A-pagination-of-filtered-results/>

## login Template

<https://ultimatedjango.com/learn-django/lessons/create-the-login-template/>

## Authentication

<https://docs.djangoproject.com/en/1.10/topics/auth/default/>

<https://djangobook.com/customizing-authentication-django/>

## Active Directory

<https://www.djangosnippets.org/snippets/501/>

<https://djangosnippets.org/snippets/901/>

### Apache authentication and Authorization

<http://www.roguelynn.com/words/apache-kerberos-for-django/>

## USING DJANGO-AUTH-LDAP WITH ACTIVE DIRECTORY

<http://www.spannerbracket.com/wordpress/?p=40>

<http://fle.github.io/combine-ldap-and-classical-authentication-in-django.html>

<http://django-auth-ldap.readthedocs.io/en/latest/example.html>

<https://www.reddit.com/r/django/comments/59qb4w/anyone_know_how_to_authenticate_with_active/>

<https://pythonhosted.org/django-auth-ldap/_static/versions/1.0.19/index.html>

<https://serverfault.com/questions/431040/django-ldap-how-to-map-description-field-to-django-group>

<https://stackoverflow.com/questions/15669161/how-to-achieve-authentication-with-django-auth-ldap>

<http://www.duanqu.tech/questions/882042/understanding-django-ldap-authentication>

<http://pythonhosted.org/django-auth-ldap/groups.html#types-of-groups>

<https://github.com/etianen/django-python3-ldap/blob/master/README.rst>

## Permission

<https://python-programming.courses/django/permission-checking-django-views/>

## Slowly Changing Dimensions

<https://en.wikipedia.org/wiki/Slowly_changing_dimension>

<https://en.wikipedia.org/wiki/Slowly_changing_dimension#Type_6_.2F_hybrid>

<http://datawarehouse4u.info/SCD-Slowly-Changing-Dimensions.html>

<http://www.kimballgroup.com/2013/02/design-tip-152-slowly-changing-dimension-types-0-4-5-6-7/>

<http://sqlblog.com/blogs/jamie_thomson/archive/2009/11/28/debunking-kimball-effective-dates.aspx>

<https://www.tutel.me/c/dba/questions/40760/how+to+snapshot+or+version+a+relational+database+when+data+changes>

<https://player.oreilly.com/videos/0636920029670>

<https://talentedmonkeys.wordpress.com/2010/05/15/temporal-data-in-a-relational-database/>

<https://www.ibm.com/support/knowledgecenter/en/SSEPGG_10.1.0/com.ibm.db2.luw.admin.dbobj.doc/doc/c0058477.html>

<https://wiki.postgresql.org/images/6/64/Fosdem20150130PostgresqlTemporal.pdf>

<https://stackoverflow.com/questions/323065/how-to-version-control-a-record-in-a-database>

## Django Simple History

<https://django-simple-history.readthedocs.io/en/latest/usage.html>

<https://stackoverflow.com/questions/tagged/django-simple-history>

<https://www.bountysource.com/teams/django-simple-history/issues?tracker_ids=106723>

<https://stackoverflow.com/questions/41527722/should-i-use-effective-date-or-start-date-and-end-date-for-historical-recording>

## Django Rest Framework

<http://www.tomchristie.com/rest-framework-2-docs/api-guide/permissions>

## DB2

<https://www.ibm.com/support/knowledgecenter/en/SSEPGG_9.7.0/com.ibm.swg.im.dbclient.python.doc/doc/t0060891.html>

# Appendix D – Deploying Code List Library

The following link outlines how to deploy Django on apache with virtualenv and mod\_wsgi:

<https://www.thecodeship.com/deployment/deploy-django-apache-virtualenv-and-mod_wsgi/>

<https://www.digitalocean.com/community/tutorials/how-to-run-django-with-mod_wsgi-and-apache-with-a-virtualenv-python-environment-on-a-debian-vps>

# Appendix E

CALL DB2XML.UpdateCodeListTable(1,102, 'david.brown', password, ?, ?);

select \* from DB2XML.MYCODES;

CALL DB2XML.GETCODELIST(103, 'david.brown', password);

# Appendix F – Testing

To run tests open the command window and navigate to the folder where manage.py is located and then run:

python manage.py test

This will create a test database and run tests within any test modules, for example tests.py and test\_views.py

# Appendix G – pgdump to generate insert statements within Postgres

Open a command window

Cd to the location where you wish to create the sql dump of insert statements

You will need to change some parameters outlined below e.g. testtable.sql this is the filename where you insert the sql statements.

Run the following command:

pg\_dump –h localhost – 5432 –U “clluser” –table=”public.clinicalcode\_codingsystem” –data-only –column-inserts –d code\_list\_library –f testtable.sql

You can now add this file to a data migration or change the script and just run it.

# Appendix H

This is not required at the moment

## Download XAMPP

Download XAMPP from the following url:

<https://www.apachefriends.org/download.html>

Double click exe and ok the warning message that appears.

Follow defaults throughout installation.

# Test Installation

Open the XAMPP control panel

In the XAMPP control panel, click on ‘Start’ under ‘Actions’ for the Apache module. This instructs XAMPP to start the Apache webserver.

If you are experiencing a problem starting the apache module then you may need to change the default port from 80 to something like 7777. You achieve this by doing the following:

Click Config on the apache section and select “Apache (httpd.conf)

Search for 80

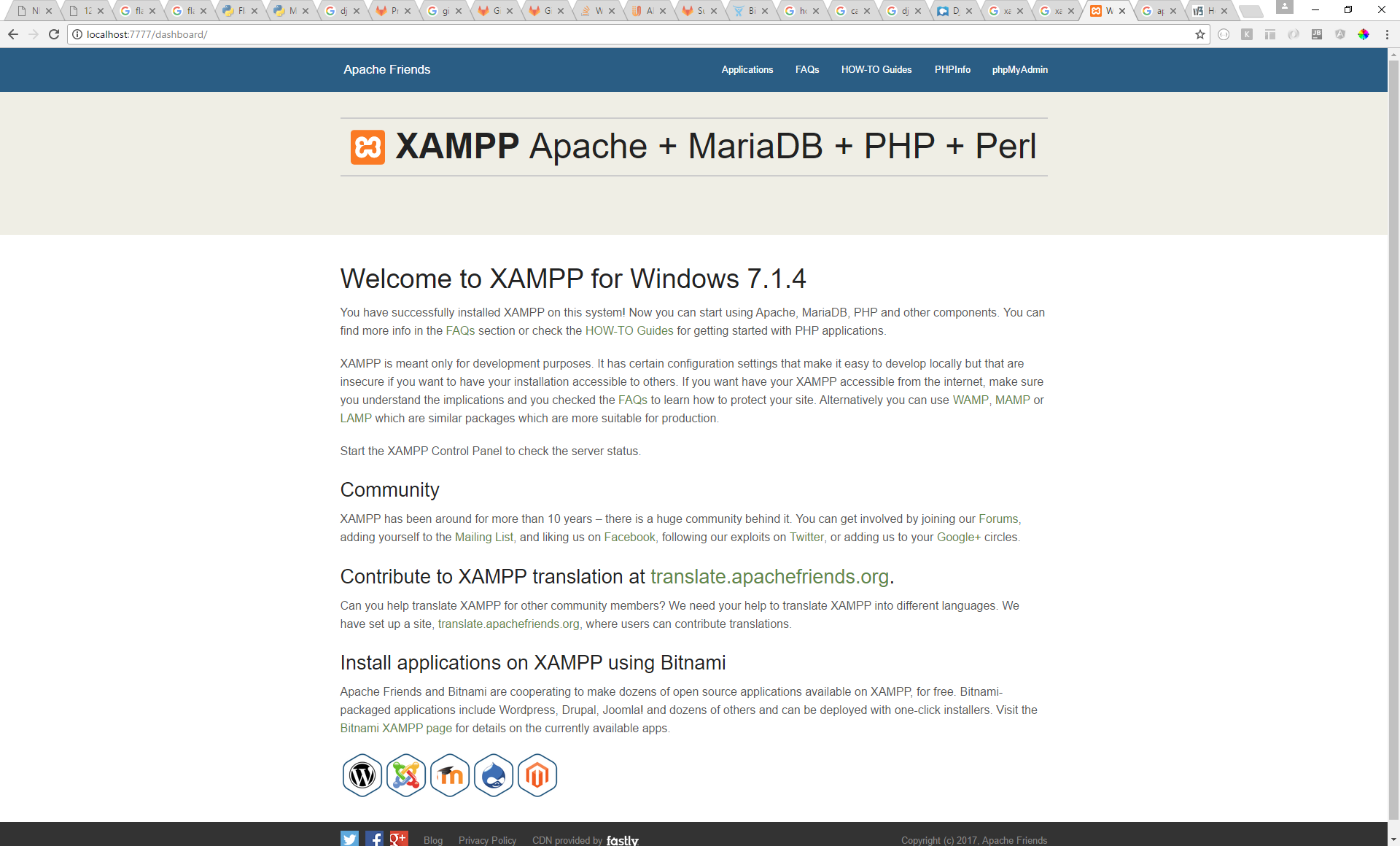
Then change “Listen 80” to be “Listen 7777”

Now click on the following link:

<http://localhost:7777>

PS: If you do not want to run XAMPP in another port, and still want to continue using the default port 80, (In our case local IIS) you need to find and stop the program or application already using the port.

If you see the following screen then you have successfully installed xampp apache.



Just a quick test:

Fire up notepad and type the following:

<?php

Echo ‘Hello world’;

?>

Save this file as ‘test.php’ in c:\xampp\htdocs\

Navigate to localhost:7777/test.php. You should see the “Hello world” message.

# Running Django on XAMPP

Install wsgi and check if everything is Ok.

Download the following file:

<https://storage.googleapis.com/google-code-archive-downloads/v2/code.google.com/modwsgi/mod_wsgi-win32-ap22py27-3.3.so>

Copy this file to the following location:

C:\xampp\apache\modules\

And change its name to mod\_wsgi.so

Now you have to locate and change apache configuration file - \xampp\apache\conf\httpd.conf

Edit it and at the end of LoadModule section add the line:

LoadModule wsgi\_module modules/mod\_wsgi.so

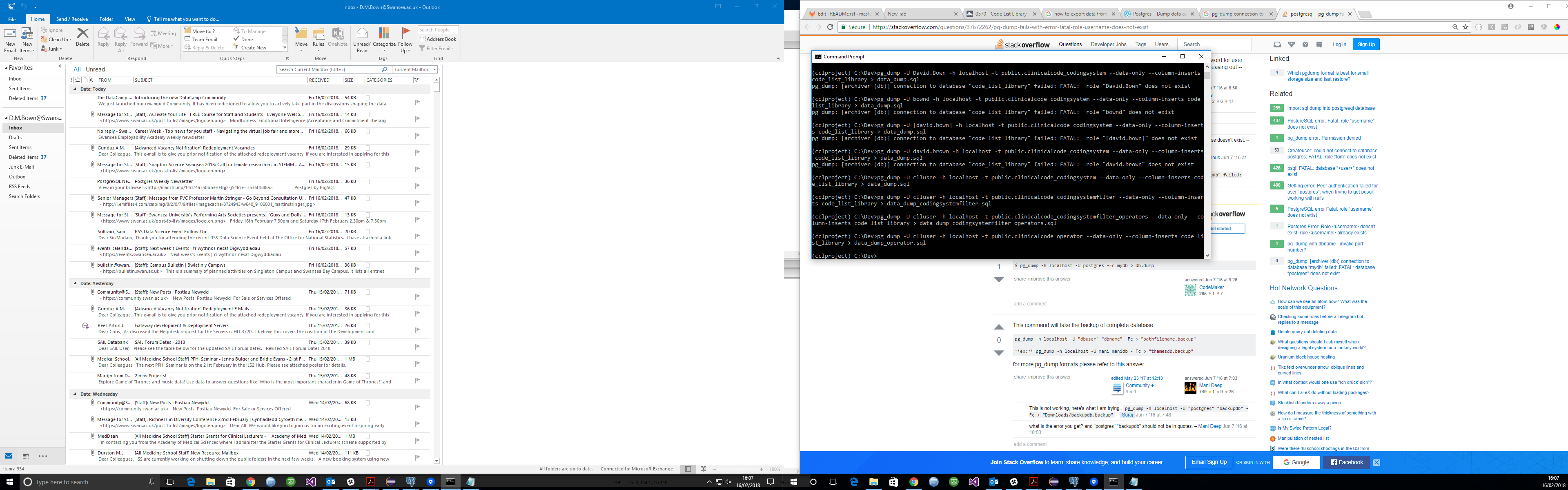
Relaunch apache using C:\xampp\xampp-control.exe

Install and run Django

Run **easy\_install django** to install Django on PC

Now Create a folder for our application:

e.g.



COPY public.clinicalcode\_codingsystem TO 'C:\dev\clinicalcode\_codingsystem.csv' DELIMITER ',' CSV HEADER;