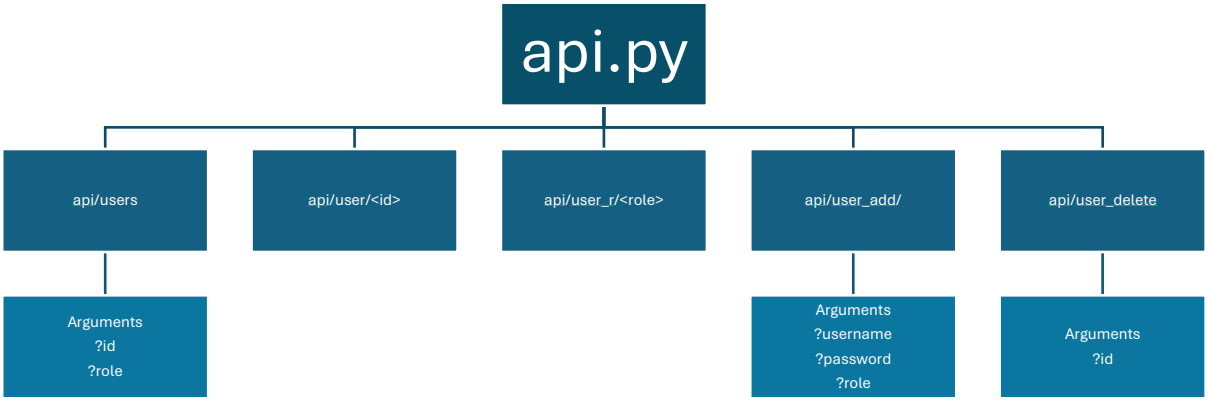
API Development

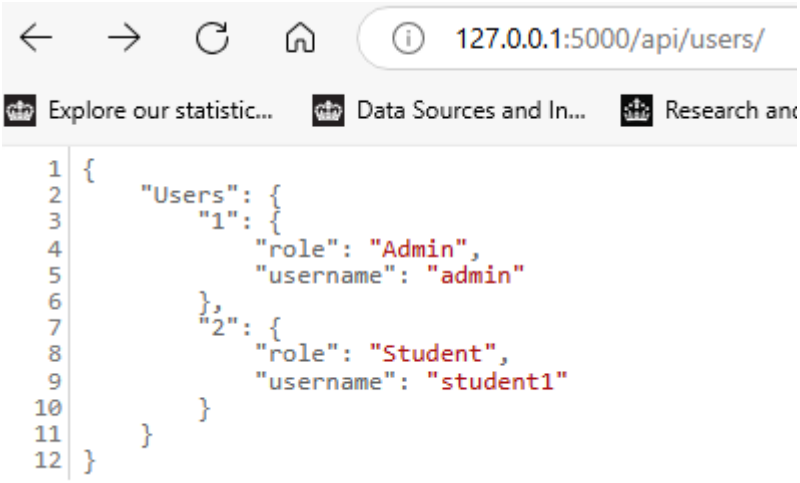
The following code has been incorporated into the School Learning Environment System to enable an API with access by a user with ‘Admin’ permissions.

The API allows the user to perform the Create, Read, Update, Delete functions on the database. Each function is performed via a web-interface with an address corresponding to a function, as shown below:

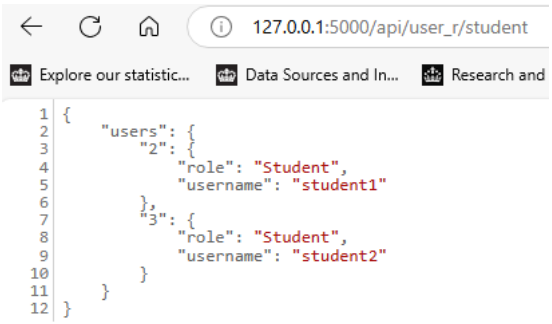
Figure 1: Ontology of API



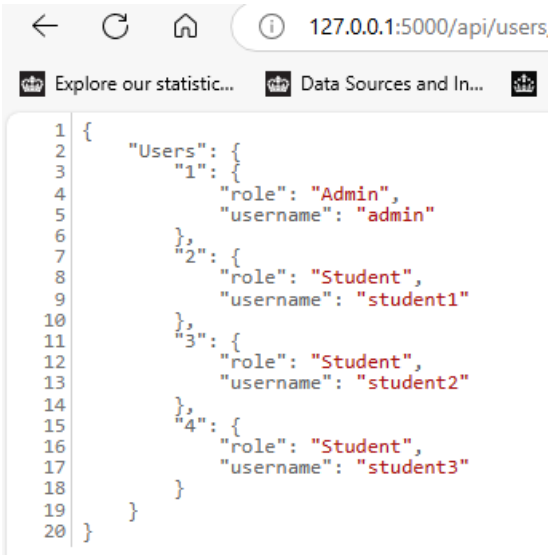
In some instances a user can pass an argument to the API to determine the appropriate response. For example, ‘/api/users’ will produce a list of all users



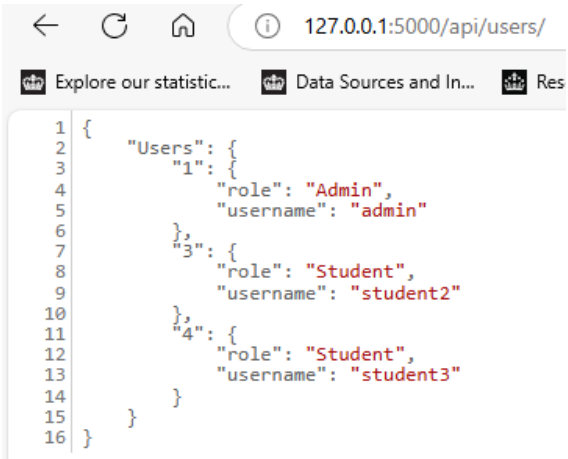
However the arguments for role or id allow the list to be filtered. For example, ‘/api/users/?role=Student will redirect the api/user_r/student page as seen below:



New users can be added, but multiple arguments have to be passed to the API with the query being concatenated. For example, /api/user_add/?username=student3&password=123&role=Student. This will add the user and redirect to the 'api/users' page:



Similarly, users can be deleted, but their id must be specified. For example to delete the user with id=2, the following command is used /api/user_delete/?id=2. Again, redirection takes place to the 'admin/users' page



Appendix 1 – api.py

```
# imports libraries
from flask import (jsonify,
                   Blueprint, redirect,  request, url_for)
# from werkzeug.exceptions import abort
from werkzeug.security import generate_password_hash
# imports functions from other modules
from flaskr.auth import secure_login
from flaskr.db import get_db


# defines the api route to be api/<name>
bp = Blueprint('api', __name__)


def list_users(users):
    # funtion to return a table of users as a dictionary list
    results = {}


    # each user record is appended to the results list
    for row in users:
        r = {'username': str(row['username']), 'role': str(row['user_role'])}
        results[str(row['id'])] = r


    return (results)


@bp.route('/api/users/')
# defined the route for the request to show all users
def users():
```

```
    # if security setting is on, the user role must be Admin to use the API
    secure_login(['Admin'])
    # queries the database to produce a all users
    db = get_db()
    users = db.execute(
        ' SELECT u.id, username, user_role'
        ' FROM user u '
    ).fetchall()

    # allows id and role to be specified as arguements in the address bar
    id = request.args.get('id')
    role = request.args.get('role')

    # if an id is specified as an arguent, a single record is returned
    if id is not None:
        return (redirect(url_for('api.user', id=id)))

    # if an id is not specified, but a role is specified
    # the records for that role will be returned
    if role is not None:
        return (redirect(url_for('api.user_role', role=role)))

    # when no arguemnents are specified, all records are shown
    results = list_users(users)
    # results are shown as a JSON list
    return jsonify({'Users': results})

@bp.route('/api/user/<int:id>')
```

```
# defines the route to show a specific user, specified by their id
def user(id):
    # if security setting is on, the user role must be Admin to use the API
    secure_login(['Admin'])

    # queries the database to show a user where the user id is 'id'
    db = get_db()
    users = db.execute(
        ' SELECT u.id, username, user_role'
        ' FROM user u '
        ' WHERE u.id=?', (id,)
    ).fetchall()
    # the results are shown as a list
    results = list_users(users)

    # the resuts are displayed as a JSON list
    return jsonify({'Users': results})

@bp.route('/api/user_r/<string:role>')
# defines the route to show a specific user, specified by their role
def user_role(role):
    # if security setting is on, the user role must be Admin to use the API
    secure_login(['Admin'])
    # queries the database to show a user where the user_role is 'role'
    db = get_db()
    users = db.execute(
        ' SELECT u.id, username, user_role'
        ' FROM user u '
```

```
        ' WHERE user_role =?', (role.title(),)
    ).fetchall()

    # the results are shown as a list
    results = list_users(users)

    # the resuts are displayed as a JSON list
    return jsonify({'users': results})

@bp.route('/api/user_add/')
# defines the route to add a user to the database
def user_add():
    # if security setting is on, the user role must be Admin to use the API
    secure_login(['Admin'])
    # user attributes are specifies as arguements in the address bar
    username = request.args.get('username')
    password = request.args.get('password')
    role = request.args.get('role')
    # the user is inserted into the database
    db = get_db()
    db.execute(
            "INSERT INTO user (username, password, user_role)\
              VALUES (?, ?, ?)",
            (username, generate_password_hash(password), role),
              )
    db.commit()
    # the list of all users is displayed
    return (redirect(url_for('api.users')))
```

```
@bp.route('/api/user_delete/')
# defines the route to add a user to the database
def user_delete():
    # if security setting is on, the user role must be Admin to use the API
    secure_login(['Admin'])
    # the id of the user to be deleted is specified
    # as an arguement in the address bar
    id = request.args.get('id')

    # the user is deleted from the database
    db = get_db()
    db.execute('DELETE FROM user WHERE id = ?', (id,))
    db.commit()
    # the list of all users is displayed
    return (redirect(url_for('api.users')))
```