University of Leeds **School of Computer Science**

COMP3011, 2024-2025

Web Services and Web Data

A RESTful API for Rating Professors

By

Saud Abu Khodair

201553433 fy21sta@leeds.ac.uk

Date: 3/10/2025

1. The Database

For the database of this service, I've decided to use the user model that comes with Django's template code and to work from there.

- User Table:
    - Id: primary key
    - Password: char field, hashed
    - Is super user: if the user is admin
    - User name: char field
    - Email: char field
- Professors Table:
    - Id: primary key
    - Name: char field
    - Code: unique, char field
- Module Table:
    - Id: primary key
    - Name: char field
    - Code: unique, char field
    - Year: integer
    - Semester: integer
    - Professors: foreign key (many to many relationship)
- Ratings Table:
    - Id: primary key
    - User Id: foreign key
    - Professor Id: foreign key
    - Module: foreign key
    - Year: integer
    - Semester: integer
    - Rating: integer

Relationships:

Users can submit multiple ratings

Professors can have multiple ratings

Modules can have multiple ratings

Professors and modules have a many to many relationship due to the rating table, a professor can teach many modules and a module can have many professors

Each user can only submit one rating per professor per module per year per semester, which is enforced by class Meta: unique_together

2. The API

The API gives the user endpoints for user authentication, professor and module listing, and rating submissions. It follows RESTful design principles and communicates via JSON-formatted data.

Register a new user:

URL: POST /users/

JSON: username, email, password

Response(201 Created): id, username, email

Possible status code: 201, 400, 409

Purpose: make a new user

Login:

URL: POST /token-login/

JSON: username, password

Response(200 OK): token

Possible status code: 200, 400, 401

Purpose: get an authentication token to use the api

List professors:

URL: GET /professors/

Response(200 OK): id, name, code, avg rating, stars

Possible status code: 200, 500

Purpose: gives you all professors in json

List modules:

URL: GET /modules/

Response(200 OK): name, code, year, semester, professors

Possible status code: 200, 500

Purpose: gives you all modules in json

Rating a professor:

URL: POST /ratings/

JSON: professor code, module code, year, semester, rating

Response(201 Created): id, year, semester, rating, created at, user, professor, module

Possible status code: 201, 400, 403, 409

Purpose: with the professor code, module code, year and semester, you can rate a professors work.

Get average rating of professor in a certain module:

URL: GET /professors/<prof-code>/modules/<mod-code>/

Response(200 OK): professor, module, avg rating, stars

Possible status code: 200, 404

Purpose: get averaged out rating of professor in a specific module

Logout a user:

URL: Post /token-logout/

Response(200 OK): user logged out

Possible status code: 200, 401

Purpose: to log out the user

3. The Client

The client is a command line application that allows the user to interact with the api.

It provides the ability to register, login, rate, get professors and modules. The client is implemented using the requests library for api communications.

The Client follows a menu-based system, where users enter commands that are processed dynamically. Commands such as register, login, logout, list, view, average, and rate correspond to api endpoints and trigger appropriate http requests. The client stores authentication tokens upon login and includes them in subsequent requests for protected routes.

For robustness, the client incorporates the following error-handling mechanisms:

- Connection handling: using a try: except: we can catch a request.exceptions.connectionerror if the API is not available without the client crashing.
- Invalid URL handling: if provided with a URL argument during registration or loging, we check if its a valid http link by checking if it has a http:// in the beginning.
- User input validation: commands that require arguments check for missing or incorrect inputs, usually by splitting, stripping, and checking the length of the args.

The client also checks and responds accordingly to http response codes such as

200: OK, 201: created

4. Using the client

The client is a command line application that lets the user interact with the API. Below are the instructions needed to to setup and install teh dependencies that the client relies on.

Requirements:

The only requirement that the client needs is “pip install requests” to deal with http requests and sessions. Alternatively since both the api and client are both developed within the same python virtual environment, you can use “pip install -r requirements.txt” to set everything up, the txt file will be provided in the zip file.

To run the client use “python client.py”, once prompted with “enter command:” here are the following commands the client accepts.

- register(url): register a new user, it will ask you to type in a username, email, password

- login(url): login with a user and password, by default the client uses the BASE_URL which is hard coded into the file for convenience, but if login is given a valid URL as an argument, it will try to log in to that instead.
- logout: logs out from the using the BASE_URL, since the base url is a global variable itll save what url you are using, weather a new one via the login arguments or the default one.
- list: list all the modules in the db
- view: view all the professors and their ratings in the db
- average (professor_code, module_code): get a professors average rating in a certain module.
- rate (professor_code, module_code, year, semester, rating): submit a professor rating
- help: prints out all valid commands
- exit: exits the programme