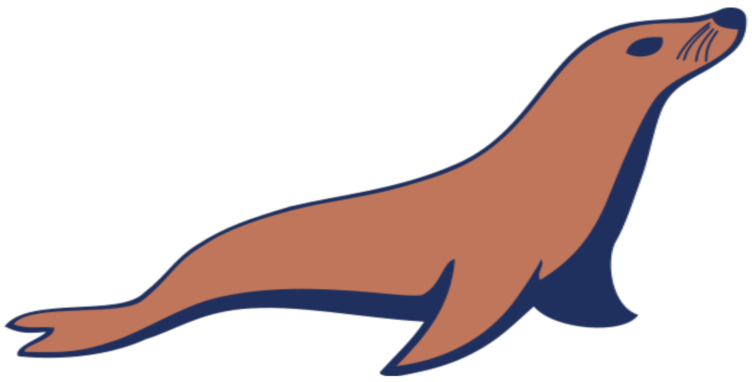
Data Modelling Tools

AUTM08016

Topic 7
Front-end Application Development



Dr Diarmuid Ó Briain
Version 1.0 [01 January 2024]

TUS
Ollscoil Teicneolaíochta na Sionainne:
Lár Tíre, An tIarthar Láir
Technological University of the Shannon:
Midlands Midwest

Dr Diarmuid Ó Briain

CEng, FIEI

CISSP®

Linux Version

```
~$ lsb_release -a | grep Description
Description:   Ubuntu 22.04.3 LTS
```

Apache2 Version

```
~$ apache2 -v
Server version: Apache/2.4.52 (Ubuntu)
Server built:   2023-10-26T13:44:44
```

MariaDB Version

```
~$ mariadb --version
mariadb  Ver 15.1 Distrib 10.6.12-MariaDB, for debian-linux-gnu
(x86_64) using  EditLine wrapper
```

python version

```
~$ python3 --version
Python 3.10.12
```

Table of Contents

Table of Figures

1. Introduction

To this point the topics have focused on the database and the backend processes responsible for processing data, handling logic, and communicating with external systems. This topic considered a frontend, also known as the presentation layer, which is the part of the application that users interact with directly. It comprises the Graphical User Interface (GUI), the visual elements that allow users to navigate, input data, and view outputs. The connection between the frontend and backend is crucial for enabling data exchange and interaction. When a user performs an action on the frontend, such as submitting a form or clicking a button, this event triggers a request to the backend. The backend then retrieves or modifies data from the MariaDB database, processes it, and sends the updated information back to the frontend for display.

To create a dynamic frontend that interacts with the MariaDB database, various web frameworks and libraries can be employed. Flask, Django, and Web.py are popular options that provide a structured framework for building web applications. By combining Python's versatility and MariaDB's robust data management capabilities, powerful and scalable web applications can be created that seamlessly deliver dynamic content to users. Python's expressiveness empowers the backend to handle complex logic and data manipulation, while MariaDB's efficiency ensures efficient storage and retrieval of information.

1.1 Objectives

By the end of this topic the learner will be able to

- Develop a front-end that interface to manage a back-end database.

2. MariaDB Connector/Python

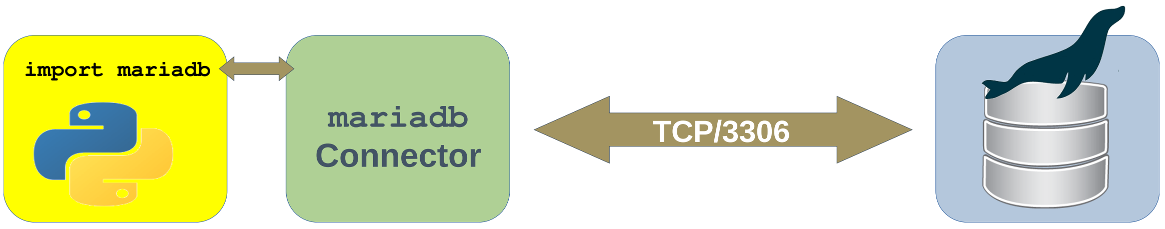


Figure 1: MariaDB Connector/Python

```
(.venv)~$ sudo python3 -m pip install mariadb
```

The MariaDB connector/Python permits the use of Python to manage data stored in MariaDB Platform. As in Figure 2, a python program that has imported the python **`mariadb`** module has access to a connection class with the following methods.

`connect()`: Establishes a connection to a MariaDB database server and returns a connection object.

`cursor()`: Returns a new cursor object for the current connection.
This keeps track of where an operation is taking place in the database.

`close()`: Closes the connection.

```
~$ ./basic_query.py
#!/usr/bin/env python3

import mariadb

query = ("SELECT * FROM EngProject;")
conn = mariadb.connect(user="enguser", password="engpass",
                       Host="127.0.0.1", port=3306,
                       database="Eng")
cur = conn.cursor()
cur.execute(query)
print(cur.fetchall())
cur.close()
conn.close()
```

Figure 2: Basic query via MariaDB Connector/Python

```
~$ ./basic_query.py [(1, 'alovelace', 'Ada', 'Lovelace',
'ada@lovelace.com', 'Programmer'), (2, 'lmenabrea', 'Luigi',
'Menabrea', 'luigi@menabrea.it', 'Politician'), (3, 'equinn',
'Edel', 'Quinn', 'edel@quinn.net', 'Nurse'), (4, 'vcunnane',
'Vincent', 'Cunnane', 'vc@tus.ie', 'Professor')]
```

2.1 Default Data Generator

The following Python program will use the MariaDB Connector/Python to populate the database with sample data. It reads in the configuration information from the, **`conf.yml`**, YAML file, requests the database password and if it is correctly supplied, it will populate the database.

```
~$ sudo apt install tree

~$ tree data_gen
data_gen
├── conf.yml
└── default_dbsetup.py
```

The **`conf.yml`** supplies the basic configuration information to the main program. This is in **`Yet Another Markup Language (YAML)`** format and is read by the main program using the **`pyyaml`** module.

```
~$ cd data_gen
data_gen~$ cat conf.yml
---
# Configuration

user: enguser          # SQL Username
host: 127.0.0.1        # SQL Database IP address
port: 3306             # SQL TCP port
database: Eng          # SQL Database
table: EngProject      # SQL table name
...
```

Consider snippets from the main program, **`default_dbsetup.py`**. The configuration information is read from the **`conf.yml`** file.

```
# // Get the configuration information from YAML //
with open(f"{__dir__}/conf.yml", "r") as fh:
    try:
        conf = yaml.safe_load(fh)
        table = conf.pop("table")
        time = conf.pop("time")
    except yaml.YAMLError as err:
        print(f"Error: {err}", file=sys.stderr)
        sys.exit(1)
```

Adding in the following lines after this section produces this output to demonstrate.

```
print(f"\ntable: {table}\nconf: {conf}")
exit()
```

```
data_gen~$ ./default_dbsetup.py
table: EngProject
conf: {'user': 'enguser', 'host': '127.0.0.1', 'port': 3306, 'database': 'Eng'}
```

Remove these added lines again.

Using the **`mariadb`** module a connection is made to the database.

```
# // Connect to the MariaDB database //
try:
    conn = mariadb.connect(
        user=conf["user"],
        password=conf["password"],
        host=conf["host"],
        port=conf["port"],
        database=conf["database"])
    cur = conn.cursor()
    print(f"\nConnected to the {conf['database']} database\n")

except mariadb.Error as e:
    print(f"\nError connecting to MariaDB Platform: {e}")
    sys.exit(1)
```

Existing tables are deleted and new tables created.

```
# // Delete the current database tables in MariaDB //
for key in db_tables.keys():
        query = f"DROP TABLE IF EXISTS Eng.{key};"
        cur.execute(query)
        print(query)

# // Create tables //
for query in db_tables.values():
        cur.execute(query)
        print(query)

# // Populate tables with some data //
for key, value in db_data.items():
        for str_ in value:
                query = f"INSERT INTO {key} VALUES ({str_})"
                cur.execute(query)
                conn.commit()
                print(query)
```

Connection is closed to the cursor and the database.

```
# // Close connection to MariaDB //
cur.close()
conn.close()
```

```
~$ cd data_gen
data_gen~$ ./default_dbsetup.py

Default engCORE Database generator
----------------------------------

This program drops existing 'EngProject' and 'EngHobbies'
tables from the 'Eng' database, creates net tables and
populates them with some sample data.

Enter the password to access database: engpass

Connected to the Eng database

DROP TABLE IF EXISTS Eng.EngProject;
DROP TABLE IF EXISTS Eng.EngHobbies;
CREATE TABLE EngProject (Student_no INT NOT NULL, Username TEXT NULL,
FirstName TEXT NULL, LastName TEXT NULL, Email TEXT NULL, Role TEXT
NULL, PRIMARY KEY (Student_no));
CREATE TABLE EngHobbies (Student_no INT NOT NULL, Hobbies TEXT NULL,
PRIMARY KEY (Student_no));
INSERT INTO EngProject VALUES (000000, 'cbabage', 'Charles',
'Babbage', 'charles@babbage.com', 'Hardware')
INSERT INTO EngProject VALUES (000001, 'alovelace', 'Ada',
'Lovelace', 'ada@lovelace.com', 'Programmer')
INSERT INTO EngProject VALUES (000002, 'lmenabrea', 'Luigi',
'Menabrea', 'luigi@menabrea.it', 'Politician')
INSERT INTO EngHobbies VALUES (000000, 'cricket, cards')
INSERT INTO EngHobbies VALUES (000001, 'camogie, horses')
INSERT INTO EngHobbies VALUES (000002, 'soccer, pasta')

Database Eng has now been populated.
```

3. Simple MariaDB Connector module

Consider the file **`mariadb_conn.py`** in the files for this topic. Copy it to the python3 path for the VM.

```
~$ python3
Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] on
linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import mariadb_conn
>>> help (mariadb_conn)

Help on module mariadb_conn:

NAME
    mariadb_conn - MariaDB connector program for SQL MasterClass

FUNCTIONS
    db_connect(**kwargs)
        MariaDB Connection

    db_delete(conn, cur, query)
        MariaDB DELETE Query

    db_insert(conn, cur, query)
        MariaDB INSERT Query

    db_select(cur, table)
        MariaDB SELECT Query

    db_update(conn, cur, query)
        MariaDB UPDATE Query

FILE
    /usr/lib/python3/dist-packages/mariadb_conn.py
```

Figure 3: mariadb connector module

It is important to gain an understanding of what this module does as it is used in the next topic. As illustrated in Figure 3, **`mariadb_conn.py`**, has five functions that simplifies the process of connecting to a database, selecting data from a database, inserting data into the database and deleting data from a database.

The program incorporates a test feature to demonstrate it is working against the database that has just been developed in this topic. This can be seen in Figure 4.

```
~$ ./mariadb_conn.py
Connected to the Eng Database
1. SELECT      2. INSERT      3. DELETE      q. QUIT      Test What? : 1

Successfully read from the EngProject Table
[('Student_no', 'Username', 'FirstName', 'LastName', 'Email', 'Role'),
 (0, 'cbabage', 'Charles', 'Babbage', 'charles@babbage.com', 'Hardware'),
 (1, 'alovelace', 'Ada', 'Lovelace', 'ada@lovelace.com', 'Programmer'),
 (2, 'lmenabrea', 'Luigi', 'Menabrea', 'luigi@menabrea.it', 'Politician')]

1. SELECT      2. INSERT      3. DELETE      q. QUIT      Test What? : 2
query: ('EngProject', (3, 'ddiddly', 'Diddly', 'Dee', 'dee@diddly.com',
'Programmer'))
Successfully inserted into the EngProject Table

Successfully read from the EngProject Table
[('Student_no', 'Username', 'FirstName', 'LastName', 'Email', 'Role'),
 (0, 'cbabage', 'Charles', 'Babbage', 'charles@babbage.com', 'Hardware'),
 (1, 'alovelace', 'Ada', 'Lovelace', 'ada@lovelace.com', 'Programmer'),
 (2, 'lmenabrea', 'Luigi', 'Menabrea', 'luigi@menabrea.it', 'Politician'),
 (3, 'ddiddly', 'Diddly', 'Dee', 'dee@diddly.com', 'Programmer')]

1. SELECT      2. INSERT      3. DELETE      q. QUIT      Test What? : 3
Successfully deleted from the EngProject Table

Successfully read from the EngProject Table
[('Student_no', 'Username', 'FirstName', 'LastName', 'Email', 'Role'),
 (0, 'cbabage', 'Charles', 'Babbage', 'charles@babbage.com', 'Hardware'),
 (1, 'alovelace', 'Ada', 'Lovelace', 'ada@lovelace.com', 'Programmer'),
 (2, 'lmenabrea', 'Luigi', 'Menabrea', 'luigi@menabrea.it', 'Politician')]

1. SELECT      2. INSERT      3. DELETE      q. QUIT      Test What? : Q

Testing completed!!
```

Figure 4: Testing mariadb_conn.py

4. Custom Database interface

4.1 Python Virtual Environment

Python has a feature called virtual environments (**`venv`**). These are self-contained directory trees that contains a Python installation for a particular version of Python, plus any required additional packages. Different applications can then use different virtual environments. This prevents conditions where one application requires a particular version of Python or a Python module that is different to other applications. It also offers a good place to store the module **`mariadb_conn.py`** created for this program as it provides a consistent path.

This module is stored in the Python virtual environment **`sys.path`**, so it is available to the web app program **`init.py`** that will be explained later in this document. This path can be identified via the Python interactive interpreter as follows in Figure 5:

Install the Virtual Environment feature.

```
~$ sudo apt install -y python3-venv
```

Create a Virtual environment in the directory **`~/.venv.`**

```
~$ python3 -m venv ~/.venv
```

Activate the virtual Environment.

```
~$ source ~/.venv/bin/activate
(.venv)~$ python3
Python 3.10.12 (main, Nov 20 2023, 15:14:05) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
Type "help", "copyright", "credits" or "license" for more information.
```

Find the path to the modules in the Virtual Environment.

```
>>> import sys

>>> sys.path

['', '/usr/lib/python310.zip', '/usr/lib/python3.10',
'/usr/lib/python3.10/lib-dynload',
'/home/ada/.venv/lib/python3.10/site-packages']

>>> quit()

~$
```

Figure 5: Python sys.path

4.2 Virtual Environment modules

The following three Python modules are installed by default when the virtual environment is created.

```
(.venv)~$ python3 -m pip list
Package       Version
------------- -------
pip           20.3.4
pkg-resources 0.0.0
setuptools    44.1.1
```

Install the following Python modules.

```
(.venv)~$ python -m pip install flask jinja2 mariadb pyyaml

(.venv)~$ python3 -m pip list
Package      Version
------------ -------
blinker      1.7.0
click        8.1.7
Flask        3.0.0
itsdangerous 2.1.2
Jinja2       3.1.2
mariadb      1.1.8
MarkupSafe   2.1.3
packaging    23.2
pip          22.0.2
PyYAML       6.0.1
setuptools   59.6.0
Werkzeug     3.0.1
```

The directory where these packages are installed is important, please note it.

```
~$ ls ~/.venv/lib/python3.10/site-packages/
blinker                     jinja2                    pkg_resources
blinker-1.7.0.dist-info     Jinja2-3.1.2.dist-info    PyYAML-
6.0.1.dist-info             click                     mariadb
setuptools                  click-8.1.7.dist-info     mariadb-
1.1.8.dist-info     setuptools-59.6.0.dist-info
_distutils_hack               markupsafe              werkzeug
distutils-precedence.pth      MarkupSafe-2.1.3.dist-info
werkzeug-3.0.1.dist-info      flask
packaging                     _yaml
flask-3.0.0.dist-info         packaging-23.2.dist-info    yaml
itsdangerous                  pip
itsdangerous-2.1.2.dist-info  pip-22.0.2.dist-info
```

4.3 How the Database connector module works

This program supplies functions to the **`init.py`** program that can read, write or delete data from the database. The complete program is available in the associated files. This section explores the functionality through various snippets.

```
# // Connect to MariaDB Platform //
def db_connect(**kwargs):
    """MariaDB Connection"""
    try:
        conn = mariadb.connect(
            user=kwargs["user"],
            password=kwargs["password"],
            host=kwargs["host"],
            port=kwargs["port"],
            database=kwargs["database"],
        )
        print(f"Connected to the {kwargs['database']} Database\n")
    except mariadb.Error as e:
        print(f"Error connecting to MariaDB Platform: {e}")
        sys.exit(1)
    return (conn, conn.cursor())
```

Figure 6: Connect to MariaDB Database

The snippet in Figure 6 demonstrates how the program establishes a database connection with the **`mariadb`** modules **`connect()`** function. This function takes the arguments necessary to connect to the **`mariadb`** database and returns a **`mariadb.connection`** object which provides an interface for the connection to the MariaDB Server. The **`cursor()`** method on the connection object retrieve a cursor, a particular interface for interaction with the Server, such as running SQL queries and managing transactions. This is a class of type **`mariadb.connection.cursor`**.

```
# // SELECT Query to MariaDB Platform //
def db_select(cur, table):
    """MariaDB SELECT Query"""

    def db_select_query(cur, query):
        """Execute the SQL SELECT Query"""

        try:
            cur.execute(query)
            return cur.fetchall()
        except:
            print("Error: Failed to get return to SELECT query")
            sys.exit(1)

    list_ = list()
    col_query = f"SHOW COLUMNS FROM {table};"
    list_ = [tuple([x[0] for x in db_select_query(cur, col_query)])]
    main_query = f"SELECT * FROM {table};"
    list_.extend(db_select_query(cur, main_query))

    return list_
```

Figure 7: Select function

The **`db_select()`** function, in Figure 7, is called with the cursor, **`cur`**, and a query consisting of a SQL **`SELECT`** query and a table name. This function has an inner function **`db_select_query()`** which is in fact called twice, once to retrieve the table column names and a second time to retrieve the table data. These are assembled into a list of tuples and returned from where they are called.

```
# // INSERT Query to MariaDB Platform //
def db_insert(conn, cur, query):
    """MariaDB INSERT Query"""

    insert_query = f"INSERT INTO {query[1]} VALUES {query[2]};"

    try:
        cur.execute(insert_query)
        conn.commit()
        print(f"Successfully inserted into the {query[1]} Database")
        return (True, insert_query)
    except:
        print(f"Error: Could not add data to {query[1]}")
        return (False, insert_query)
```

Figure 8: Insert function

The INSERT function, **`db_insert()`**, illustrated in Figure 8, receives the connection object, **`conn`**, the cursor object, **`cur`**, and a query tuple consisting of the table name and a tuple of values. These are assembled into a query, the query is executed and committed on the connection.

```
# // DELETE Query to MariaDB Platform //
def db_delete(conn, cur, query):
    """MariaDB DELETE Query"""

    delete_query = f"DELETE FROM {query[1]} WHERE {query[2]
[0]}={query[2][1]};"

    try:
        cur.execute(delete_query)
        conn.commit()
        print(f"Successfully deleted from the {query[1]} Database")
        return True
    except:
        print(f"Error: Could not delete data from {query[1]}")
        sys.exit(1)

    return True
```

Figure 9: Delete function

In a similar way, the **`db_delete`** function, illustrated in Figure 9, given the table name, a column name and a value will delete the row from the table. Where possible it makes sense to use the **`Student_no`** as the field as this is the primary key in the database and therefore is unique.

The code after the **`if __name__ == "__main__":`** loop is only ran if the **`mariadb_conn.py`** module is ran standalone, as shown in Figure 4. This is only for testing the module, under normal operation the module is called from the **`init.py`** program and this part is ignored.

Consider the files in the **`web_main`** directory. This is the Python Flask web app for the device holding the main database.

```
(.venv)~$ tree ~/web_main
/home/ada/web_main
├── app.wsgi
├── conf.yml
├── init.py
├── README.txt
├── static
│   ├── css
│   │   └── main.css
│   └── images
│       └── TUS_White.png
├── templates
│   ├── about.html
│   ├── delete.html
│   ├── home.html
│   ├── index.html
│   ├── layout.html
│   ├── login.html
│   ├── read.html
│   └── write.html
└── tools
    ├── default_dbsetup.py
    └── mariadb_conn.py
```

Move the **`mariadb_conn.py`** module in the python virtual environment **`sys.path`** as the Apache2 webserver WSGI has been supplied with that path:

```
~$ cp ~/web_main/mariadb_conn.py ~/.venv/lib/python3.10/site-packages/
```

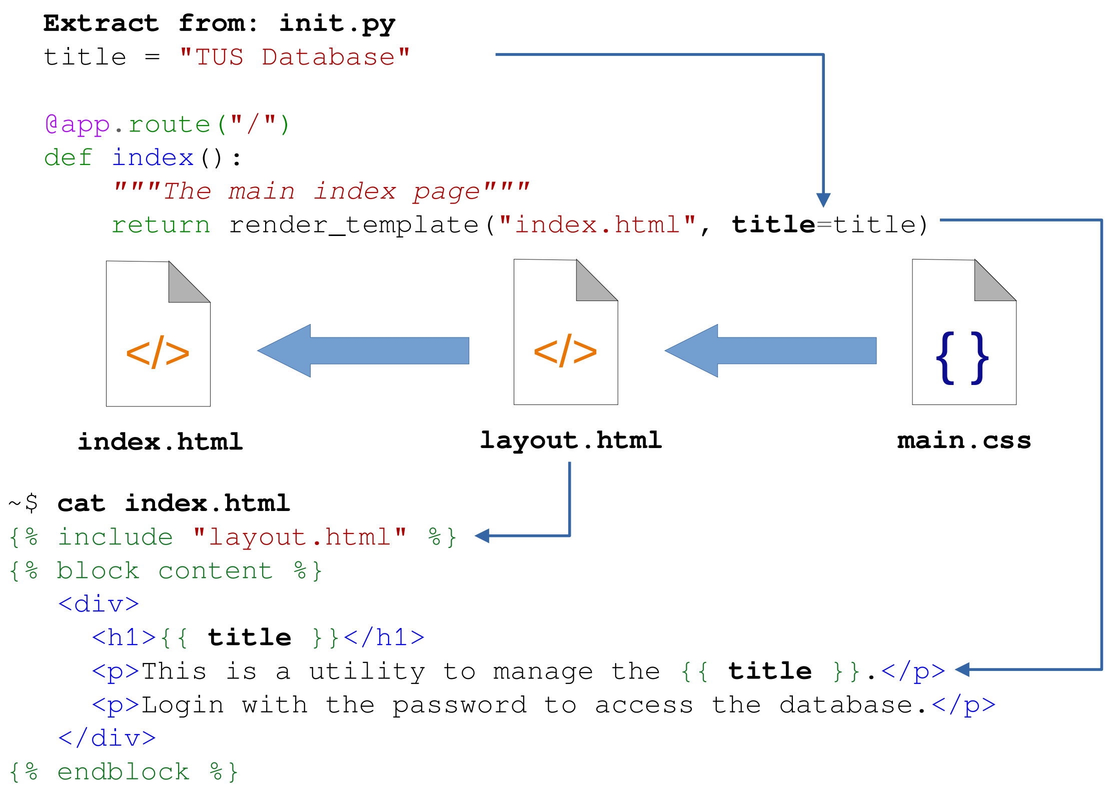
The app can be tested using a development environment before being moved to production on the Apache2 webserver.

```
(.venv)~$ python3 ~/web_main/init.py
 * Serving Flask app 'init'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production
deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 460-699-611
```

Using a browser on the same device browse to the URL indicated in Figure 11. Note the following lines appear in the terminal.

```
127.0.0.1 - - [15/Oct/2022 14:31:18] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [15/Oct/2022 14:31:18] "GET /static/css/main.css HTTP/1.1" 200 -
127.0.0.1 - - [15/Oct/2022 14:31:18] "GET /static/images/TUS_White.png HTTP/1.1" 200 -
127.0.0.1 - - [15/Oct/2022 14:31:18] "GET /favicon.ico HTTP/1.1" 404 -
```

4.4 The creation of index.html from templates



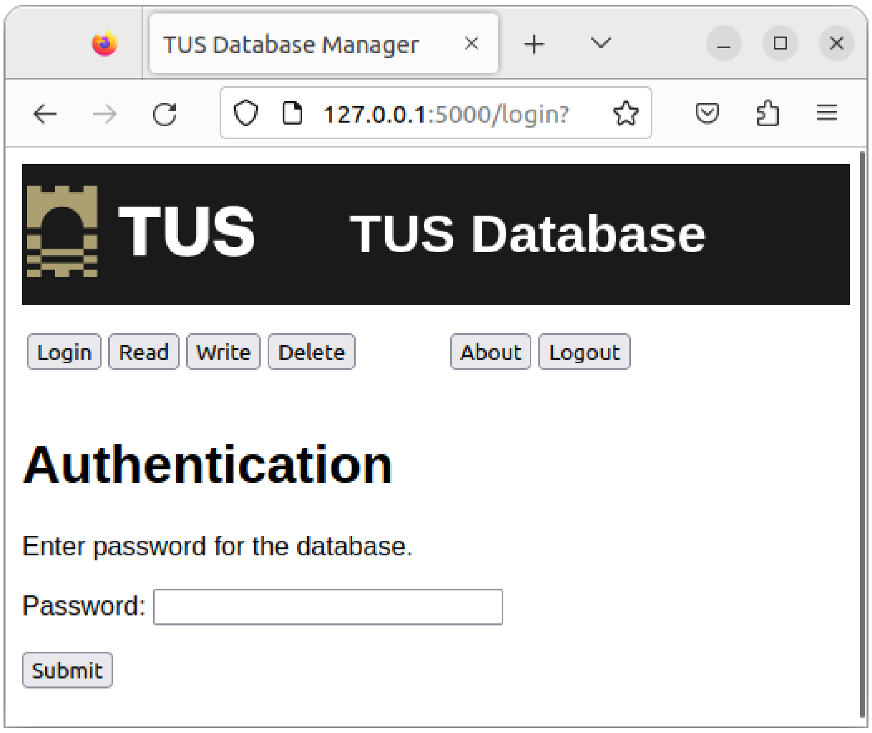
```
Extract from: init.py
title = "TUS Database"

@app.route("/")
def index():
    """The main index page"""
    return render_template("index.html", title=title)
```

```
~$ cat index.html
{% include "layout.html" %}
{% block content %}
   <div>
     <h1>{{ title }}</h1>
     <p>This is a utility to manage the {{ title }}.</p>
     <p>Login with the password to access the database.</p>
   </div>
{% endblock %}
```

Figure 10: How index.html is built

`index.html` in the templates directory is only part of the page presented. As shown in Figure 10, using Jinja2 templating, **`index.html`** includes that **`layout.html`** template which in turn uses the Cascading Style Sheet (CSS), **`static/css/main.css`**. Jinja2 allows for the embedding of **`layout.html`** at the top of **`index.html`** and allows for variable interpolation inside **`{{ }}`**. This can be seen for the string variable **`title`**. A similar process is carried out for the **`login`**, **`home`**, **`read`**, **`write`**, **`delete`** and **`about`** routes from **`init.py`**.



Authentication

Enter password for the database.

Password:

Submit

Figure 11: Browse to development webserver

```
# // Flask app //
app = Flask(__name__)

@app.route("/")
def index():
    """The main index page"""
    return render_template("index.html", title=title)
```

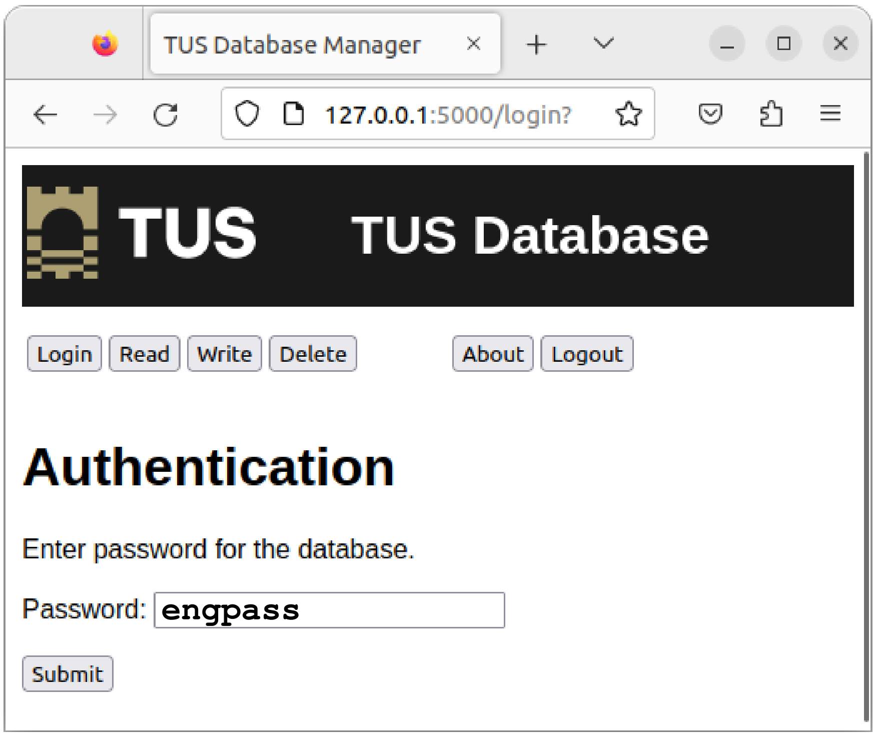
The webserver is driven by the **`init.py`** python program. When **`init.py`** is called **`Flask()`** is instantiated as **`app`**. This presents the default route which renders the **`index.html`** page.

4.5 Login to the application

Select the `Login` button and a password box is presented, which is expecting the password for the `enguser` user in the database. Without this, the application will not be able to connect to the database. Be careful not to hardcode the password for the database into the code as this is a clear security issue. The sample program stores the password in a hidden temporary file `/tmp/.pass` when using the development server. When moved to apache the webserver establishes a temporary file for its process.

```
~$ cat /tmp/.pass
engpass

~$ sudo -s
~# cat /tmp/systemd-private-aedf7ba0df264e66a949a17912810802-
apache2.service-Cgcetf/tmp/.pass
engpass
```



Authentication

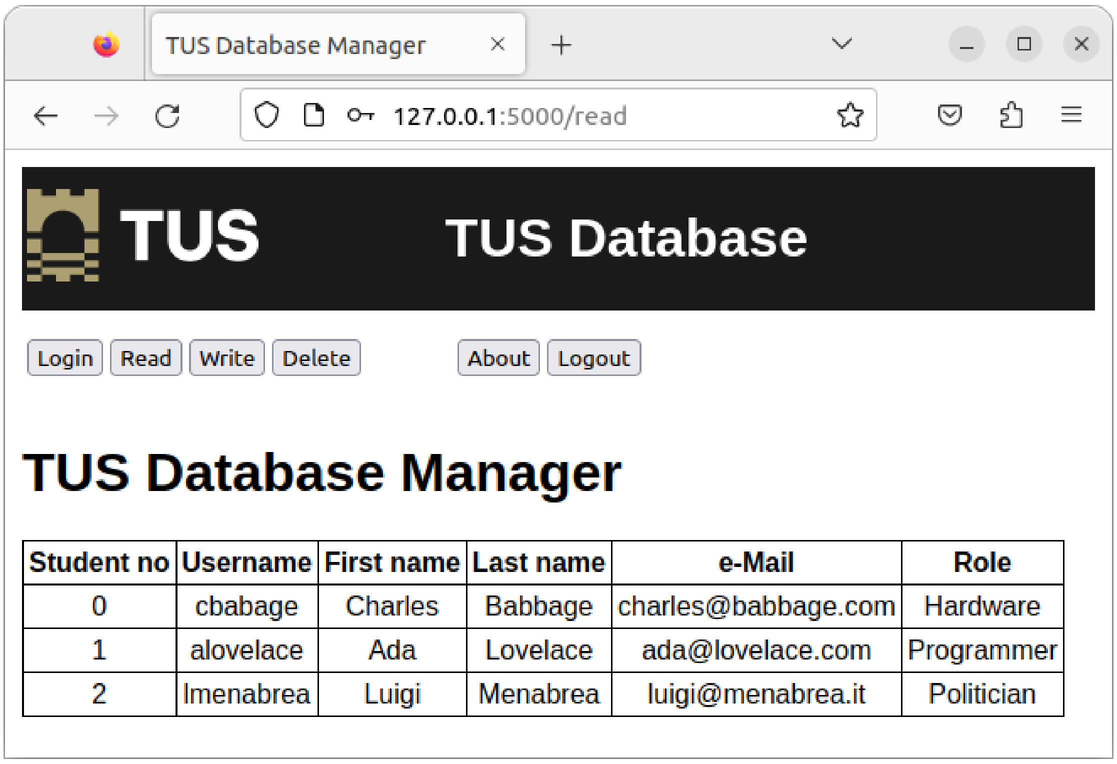
Enter password for the database.

Password: engpass

Figure 12: Root of custom application

Now the user can select Read, Write, Delete, About or Logout buttons.

4.6 Read from the database



TUS Database Manager

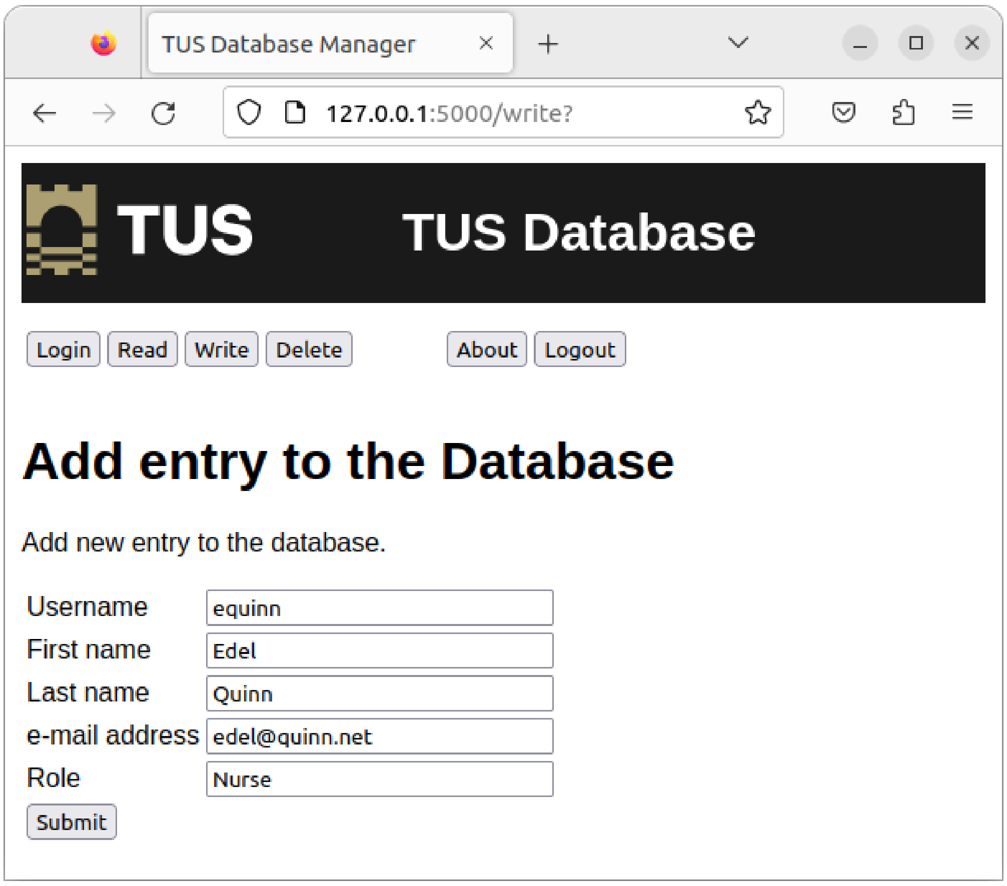
Student no	Username	First name	Last name	e-Mail	Role
0	cbabage	Charles	Babbage	charles@babbage.com	Hardware
1	alovelace	Ada	Lovelace	ada@lovelace.com	Programmer
2	lmenabrea	Luigi	Menabrea	luigi@menabrea.it	Politician

Figure 13: Read from the Database

By selecting the **`Read`** button the **`db_connect()`** followed by the **`db_select()`** functions in the **`mariadb_conn.py`** module are called. This creates a connection to the database and executes the SQL Query:

```
SELECT * FROM 'EngProject';
```

4.7 Write to the Database



Add entry to the Database

Add new entry to the database.

Username	equinn
First name	Edel
Last name	Quinn
e-mail address	edel@quinn.net
Role	Nurse

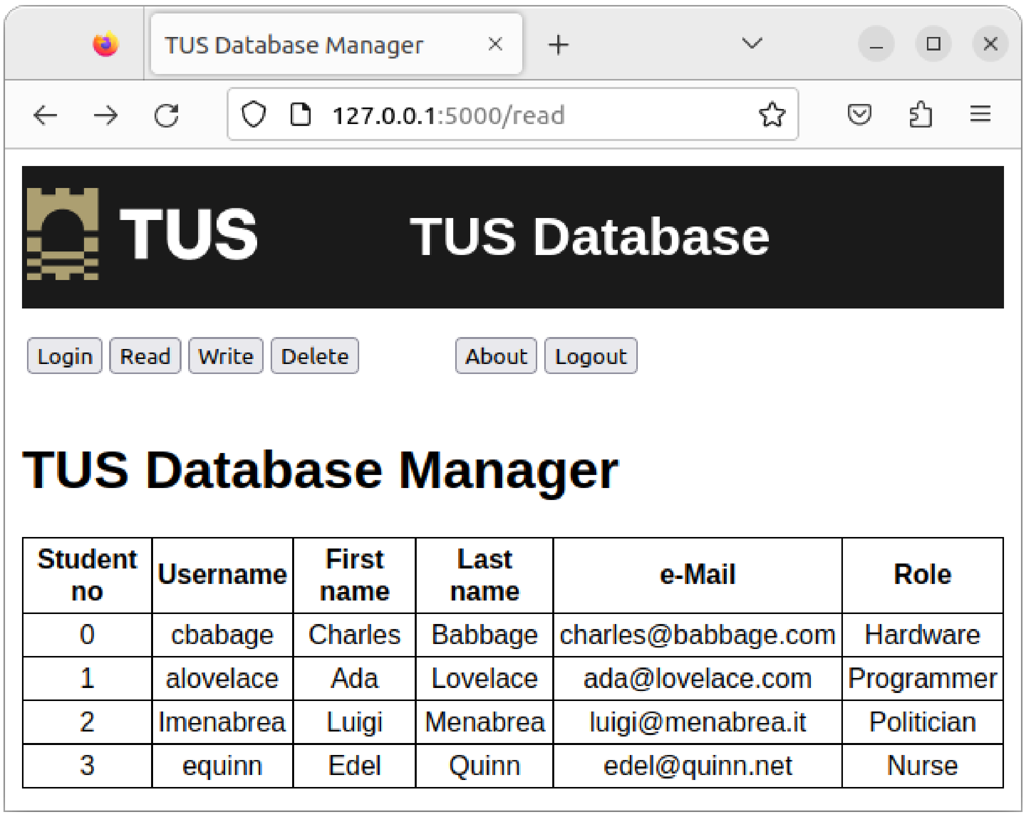
Submit

Figure 14: Write to the database

As illustrated in Figure 14, upon selecting the **`Write`** button, a form is presented, once filled and the **`Submit`** button is selected the **`db_connect()`** followed by the **`db_insert()`** functions in the **`mariadb_conn.py`** module are called. This finds the **`Student_no`** for the existing entries, sorts them and selects the next available number. It then executes the query:

```
INSERT INTO 'EngProject' VALUES(<next number>, 'equinn',
'Edel', 'Quinn','eden@quinn.net','Nurse');
```

If the **`Read`** button is selected, and as illustrated in Figure 15, the newly added user can be seen within the database.



TUS Database Manager

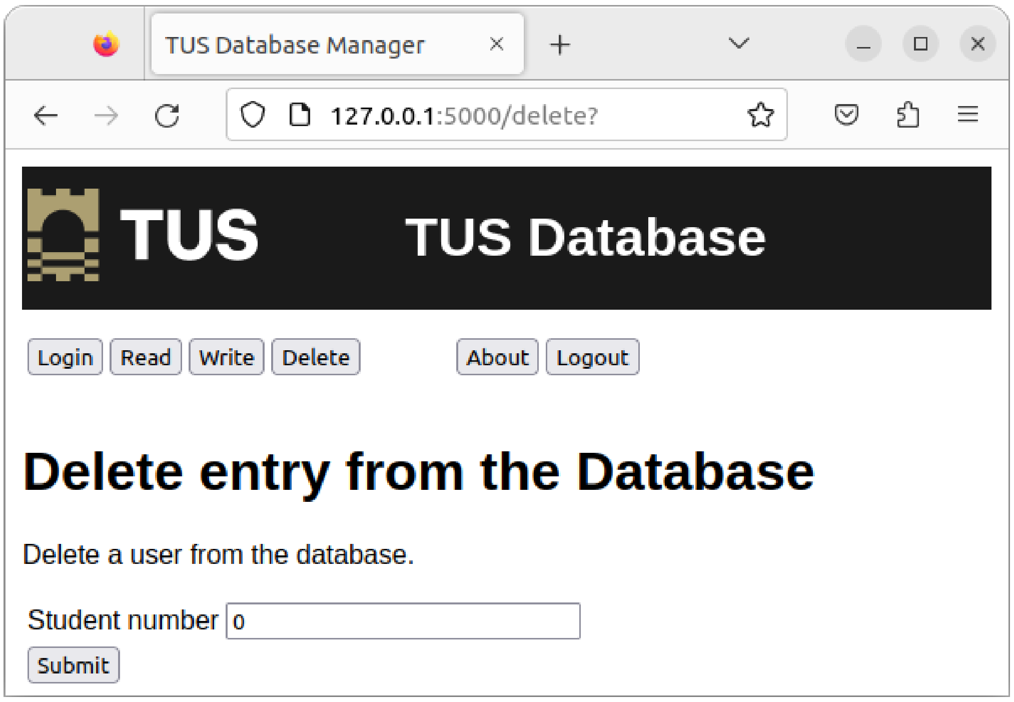
Student no	Username	First name	Last name	e-Mail	Role
0	cbabage	Charles	Babbage	charles@babbage.com	Hardware
1	alovelace	Ada	Lovelace	ada@lovelace.com	Programmer
2	lmenabrea	Luigi	Menabrea	luigi@menabrea.it	Politician
3	equinn	Edel	Quinn	edel@quinn.net	Nurse

Figure 15: View the newly added user

4.8 Delete from the Database

As shown in Figure 11, upon selecting the **`Delete`** button, a form is presented requesting a Student number for deletion. For example to delete Ada Lovelace from the database, enter her number **`0`**, and select the **`Submit`** button. The **`db_connect()`** followed by the **`db_delete()`** functions in the **`mariadb_conn.py`** module are called. This executes the query:

```
DELETE FROM EngProject WHERE 'Student_no'=0;
```



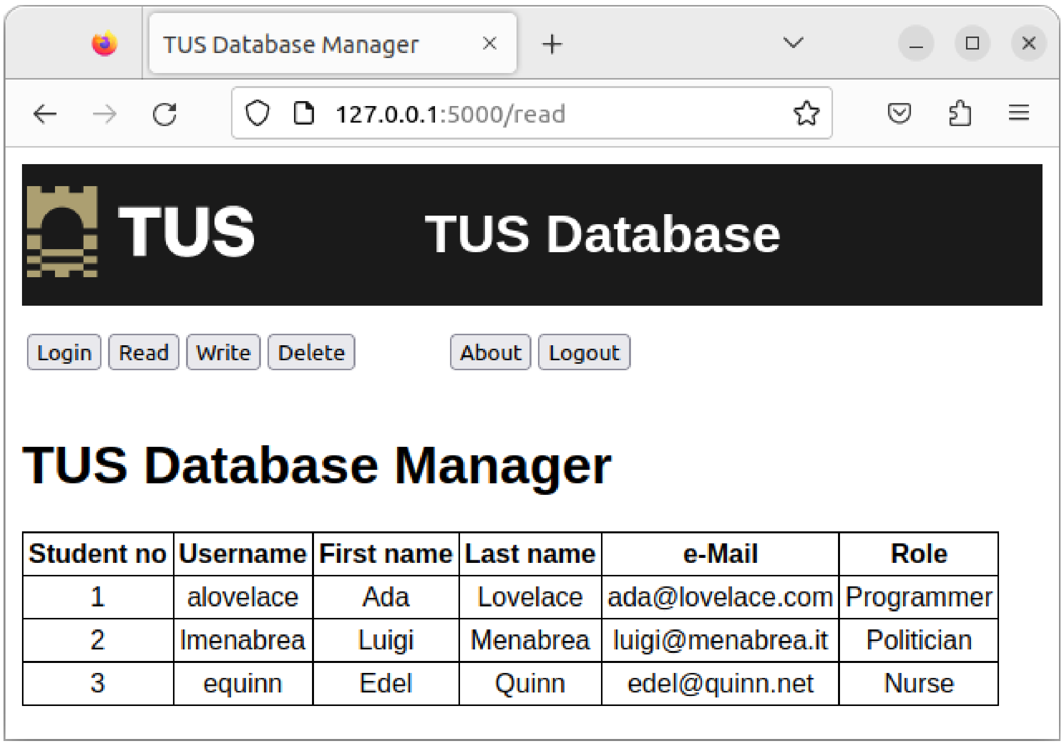
Delete entry from the Database

Delete a user from the database.

Student number 0

Figure 16: Delete from the database

Again a Read of the database confirms the delete, this is illustrated in Figure 17.



TUS Database Manager

Student no	Username	First name	Last name	e-Mail	Role
1	alovelace	Ada	Lovelace	ada@lovelace.com	Programmer
2	lmenabrea	Luigi	Menabrea	luigi@menabrea.it	Politician
3	equinn	Edel	Quinn	edel@quinn.net	Nurse

Figure 17: Read to confirm the delete

5. Moving the application into production on Apache2

Now the Python web application is running it is necessary to move it from the development server into production. Move the web app under the Apache2 server root.

```
~$ sudo cp -r ~/web_main /var/www/html/web
~$ sudo chown -R www-data: /var/www/html/web
```

Give the website user access to the group hosting the virtual environment and vicaversa.

```
~$ sudo usermod -a -G ada www-data
~$ sudo usermod -a -G www-data ada

~$ cat /etc/group | grep ^www
www-data:x:33:ada
~$ cat /etc/group | grep ^ada
ada:x:1000:www-data
```

5.1 Apache2 site configuration file

Create an Apache2 site configuration file by replacing the default file, this points to the Web Server Gateway Interface (WSGI), **app.wsgi** and the directory for the Python app.

```
~$ cd /etc/apache2/sites-available/

/etc/apache2/sites-available$ sudo mv 000-default.conf 000-default.conf.orig

/etc/apache2/sites-available$ cat <<EOM | sudo tee 000-default.conf

<VirtualHost *:80>
        ServerAdmin webmaster@localhost

        WSGIScriptAlias / /var/www/html/web/app.wsgi
        <Directory /var/www/html/web>
            Order allow,deny
            Allow from all
        </Directory>

        ErrorLog /error.log
        CustomLog /access.log combined

</VirtualHost>
EOM
```

For the python3 instance employed by the Apache2 server, the **app.wsgi** file adds to the Python path the virtual environment path created earlier as that is the source of the python modules as well as the root of the app, on the Apache2 webserver, as that is the location of the **init.py** and all its associated files. It then imports **app**, the **Flask()** instance from the **init.py** file acting as a module **init** in this instance.

```
~$ sudo cat /var/www/html/web/app.wsgi
import sys

sys.path.insert(0, "/home/ada/.venv/lib/python3.10/site-packages")
sys.path.insert(0, "/var/www/html/web")

from init import app as application
```

5.2 Install the WSGI and enable it for Apache2

Install the Apache2 WSGI library and, using a2enmod enable the library module within the apache2 configuration.

```
~$ sudo apt install libapache2-mod-wsgi-py3
~$ sudo a2enmod wsgi
Enabling module wsgi.
```

5.3 Launch Apache2 Server

Relaunch the Apache2 Server to read in the library module configuration and therefore enable the python application.

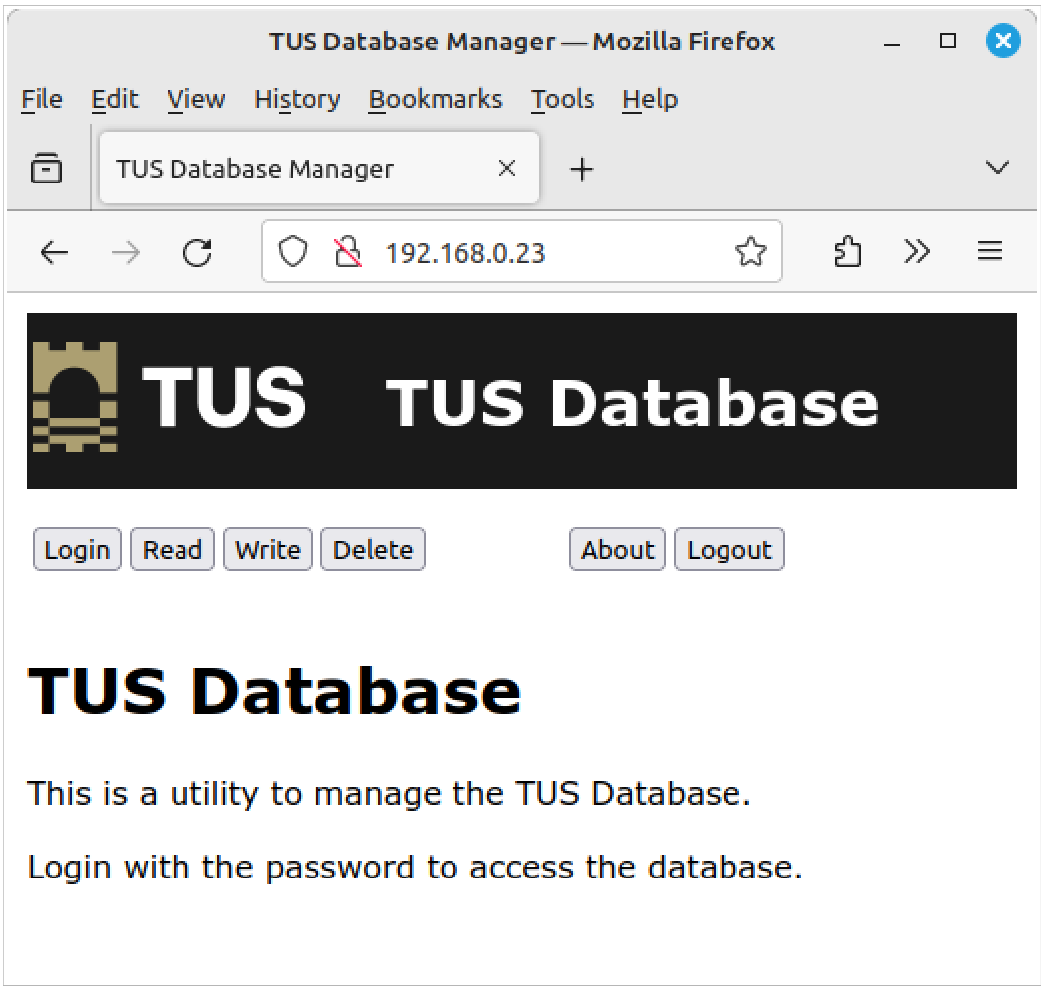
```
~$ sudo systemctl restart apache2

~$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor prese>
     Active: active (running) since Thu 2023-12-28 13:32:10 GMT; 17s ago
       Docs: https://httpd.apache.org/docs/2.4/
    Process: 7948 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SU>
   Main PID: 7953 (apache2)
      Tasks: 6 (limit: 9430)
     Memory: 29.2M
        CPU: 219ms
     CGroup: /system.slice/apache2.service
             ├─7953 /usr/sbin/apache2 -k start
             ├─7955 /usr/sbin/apache2 -k start
             ├─7956 /usr/sbin/apache2 -k start
             ├─7957 /usr/sbin/apache2 -k start
             ├─7958 /usr/sbin/apache2 -k start
             └─7959 /usr/sbin/apache2 -k start

Dec 28 13:32:10 ada-VirtualBox systemd[1]: Starting The Apache HTTP Server...
Dec 28 13:32:10 ada-VirtualBox apachectl[7952]: AH00558: apache2: Could not rel>
Dec 28 13:32:10 ada-VirtualBox systemd[1]: Started The Apache HTTP Server.
```

5.4 Test the new service in production

As illustrated in Figure 18, from another workstation, browse to the host with the Apache2 service. The TUS Database application can be accessed and operates in the same way the development one did locally.


TUS Database Manager — Mozilla Firefox
File Edit View History Bookmarks Tools Help
TUS Database Manager
192.168.0.23
TUS
TUS Database
Login Read Write Delete About Logout

TUS Database

This is a utility to manage the TUS Database.

Login with the password to access the database.

Figure 18: View service from another workstation

6. Laboratory #1

6.1 Create a custom interface to the counties database

- Create a web based interface to the database.
- Permit input of County capitals, rivers, mountains and sports.
- Do no permit removal of county information from the Counties table once entered.
- Have a mechanism to change the county sport if it is wrong.
- Document each stage.

Notes:

This page is intentionally blank