SQLite 10

GUI and SQLite

LILLEBAELT ACADEMY OF
PROFESSIONAL HIGHER EDUCATION

Author
Martin Grønholdt
mart80c7@edu.eal.dk

Monday 1 May 2017

Table of Contents

1. Introduction

This document introduces the development aspects of a customer database using SQLite as the data storage and Flask to present an interface for the user. There are many aspects of using Flask to create a web application that are not documented in the following. The Flask homepage contains many resources for learning more.

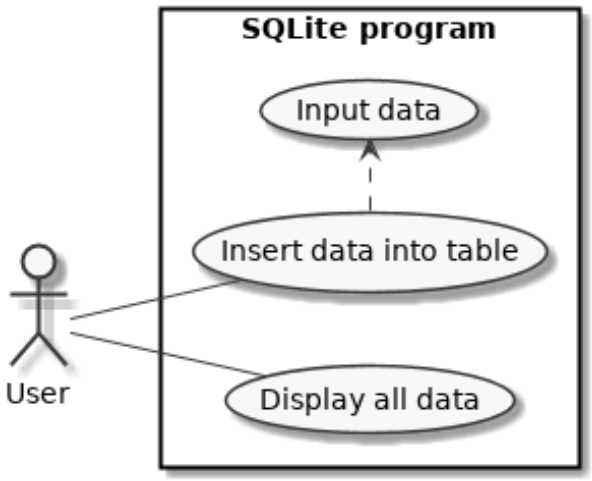
This is a bare bones implementation that would benefit from further development in these areas:

- Replace/Delete customers.
- Easier deployment (Docker, venv, k8s, …).
- Proper validation of form input.
- Proper validation of SQL queries.
- Search in the customer data.
- Encrypt traffic.
- REST API.

All hand ins for this course is available on GitHub at:
https://github.com/deadbok/eal_programming

2. Functional description

The program is a basic customer database, showing a list of all customers and add new ones, as seen in the use case diagram below.



Use case diagram

The graphical user interface has been implemented using Flask, which is a micro web framework for Python. The program act like a web server, and the user interface is accessed using a regular HTML5 compliant browser.

Using a web framework has both advantages and disadvantages in comparison with the a standard OS managed GUI components.

Advantages

- Default client/server architecture
  - No installation on the users computer, only a web browser is needed.
  - Centrally stored database.
  - Easier deployment due to single installation.
- Familiar interface using the Bootstrap CSS framework.
- User interface is build using HTML which is more accesible than a regular GUI toolkit.
- Simple to add a REST api.

Disadvantages

- Network based
  - Unencrypted traffic (use encryption).
  - Higher exposure.
- Installation is harder for non-technical users
- Ideally needs a Unix server to run.

3. Running

To run the program in development mode, run “run.py” the root of the project directory.

```
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger pin code: 147-873-485
127.0.0.1 - - [30/Apr/2017 22:12:07] "GET /add HTTP/1.1" 200 -
127.0.0.1 - - [30/Apr/2017 22:16:10] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [30/Apr/2017 22:26:40] "POST /add HTTP/1.1" 302 -
127.0.0.1 - - [30/Apr/2017 22:26:41] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [30/Apr/2017 22:27:08] "GET /add HTTP/1.1" 200 -
127.0.0.1 - - [30/Apr/2017 22:33:11] "POST /add HTTP/1.1" 302 -
127.0.0.1 - - [30/Apr/2017 22:33:11] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [30/Apr/2017 22:33:18] "GET /add HTTP/1.1" 200 -
127.0.0.1 - - [30/Apr/2017 22:34:34] "POST /add HTTP/1.1" 302 -
127.0.0.1 - - [30/Apr/2017 22:34:34] "GET / HTTP/1.1" 200 -
* Detected change in '/mnt/data/Documents/Skole/It-et/2016/programming/SQLite
10/app/views/add.py', reloading
* Restarting with stat
* Debugger is active!
* Debugger pin code: 147-873-485
```

Console output of the program running in development mode.

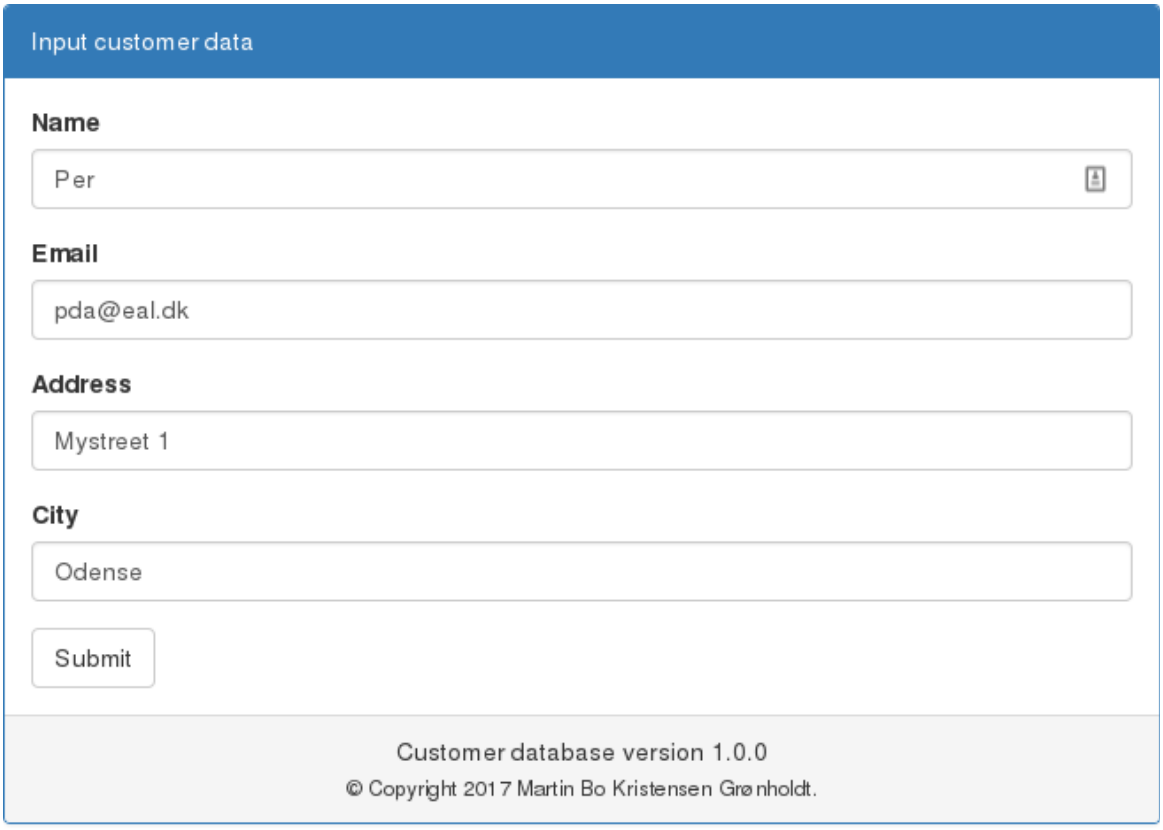
The console output shows request and responses handled by the app. Notice that because the program runs in development mode, it will reload when it detects a change in one of the source files.

4. User interface

The user interface is made using HTML and the Python template language Jinja 2. The GUI mock-ups are written in HTML and the data logic is achieved using Jinja 2. The user interface is accessed in a browser using the URL that points to the installed version, in case of the development version, this URL is http://localhost:5000/.

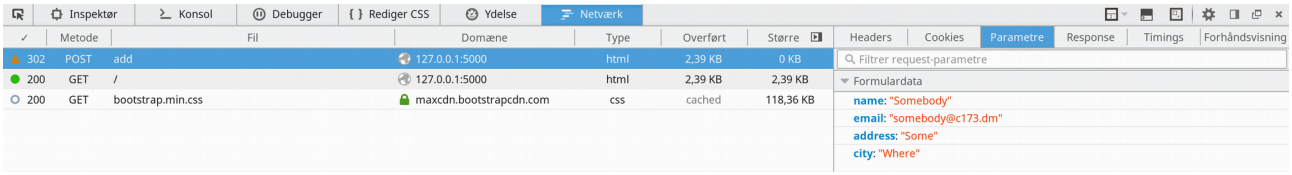
Main view with customer data and the framed add button in the upper right corner.

The main window shows the current customer records and a button to add a new customer record. The customer id is automaticity incremented by SQLite when a new customer is added.



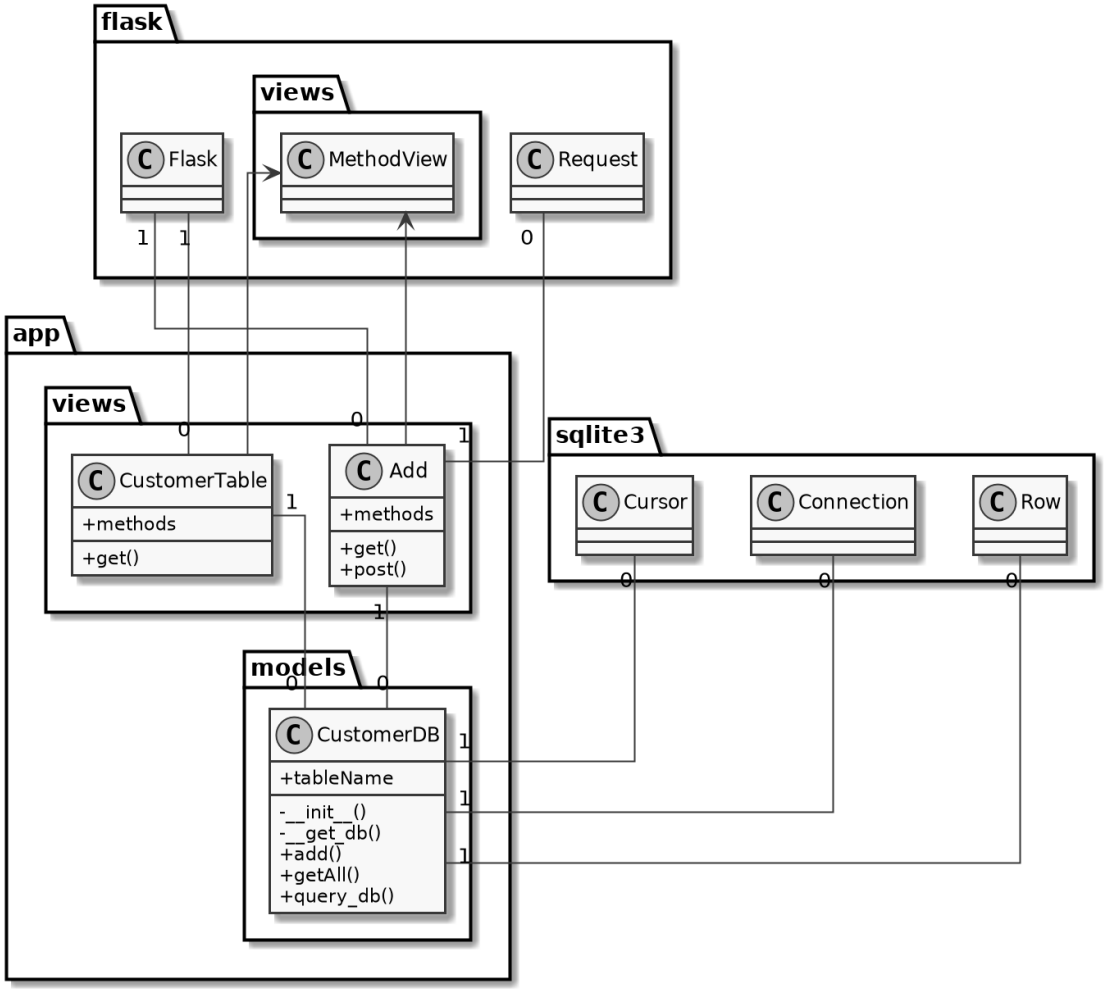
User interface for adding a new customer.

The interface to add a customer is a regular HTML form that submits a POST request back to the program. The Form does basic validation using HTML 5. Unfortunately it has proven very hard to capture a screenshot of a validation error but if a field is empty, the browser will display an error notification.



The POST request for a new customer captured in Firefox.

5. Classes



Class diagram

5.1. flask.Flask

This is the main class for any Flask application.

5.2. flask.Request

This is a Flask class containing the request data, it is used to get the form data from the POST request, when adding new customers

5.3. flask.views.MethodView

This is a Flask helper class for making views handling HTTP methods, it is inherited by the app.views.Add and app.views.CutomerTable.

5.4. sqlite3.Cursor

SQLite object used to do queries on the customer database, by the database handling code in app.models.CustomerDB.

5.5. sqlite3.Connection

SQLite object that represents the connection to the customer database, used by the database handling code in app.models.CustomerDB.

5.6. sqlite3.Row

SQLite helper class to ease working with rows in the customer database by trying to represent them as tuples, used by the database handling code in app.models.CustomerDB.

5.7. app.models.CustomerDB

Customer database handling code called by the views when they need to access or change the customer data.

5.8. app.views.Add

View to add a new customer. It renders a form for entering new customer data on GET requests and adds the data on a POST request.

5.9. app.views.CutomerTable

View to show all customers. It renders a table of all customer data on GET requests.

6. Source code

6.1. run.py:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""
Name: Main program to run the debug version of the Flask application.
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0.0 (2017-04-30)
"""
from app import APP

# Run the Flask application in debug mode.
APP.run(debug=APP.config['DEBUG'], host='0.0.0.0')
```

6.2. config.py

```
# -*- coding: utf-8 -*-
"""
Name: Configuration file for the Flask application.
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0.0 (2017-04-30)
"""

# Debugging on.
DEBUG = True
# Database file name.
DATABASE = 'customers.db'
```

6.3. app/__init__.py

```
# -*- coding: utf-8 -*-
"""
Name: Main initialisation of the Flask program.
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0.0 (2017-04-30)
"""
from flask import Flask
from app.views import Add
from app.views import CustomerTable
from flask import g

# Init app and config
APP = Flask(__name__)
APP.config.from_object('config')
CONFIG = APP.config

@APP.teardown_appcontext
def close_connection(exception):
    """
    Function to close the database connaction when the current session closes.
    """
    db = getattr(g, '_database', None)
    if db is not None:
```

```
        db.close()

# Add the / view.
APP.add_url_rule('/',
                 view_func=CustomerTable.as_view('table'),
                 methods=['GET'])
# Add the /add view.
APP.add_url_rule('/add',
                 view_func=Add.as_view('add'),
                 methods=['GET', 'POST'])
```

6.4. app/models/customerdb.py

```
# -*- coding: utf-8 -*-
"""
Name: Database abastraction code.
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0.0 (2017-04-30)
"""
import sqlite3
from flask import g
from flask import current_app

DATABASE = 'customers.db'


class CustomerDB:
    tableName = 'customerTable'

    def __init__(self):
        """
        Contrusctor that creates the database table if not there.
        """
        # Get the current tables in the database.
        db = self.__get_db()
        tables = db.execute(
            "SELECT name FROM sqlite_master WHERE type='table';")
        # Variable to indicate if the customerTable table exists
        created = False
        # Run through all table names.
        for table in tables:
            # If customerTable exists, do tell.
            if table[0] == self.tableName:
                created = True
        # Create the customerTable, if it is not there
        if not created:
            current_app.logger.debug("Creating table.")
            db.execute(
                """
                CREATE TABLE {} (
                  idCust INTEGER PRIMARY KEY,
                  name VARCHAR(100),
                  email VARCHAR(50),
                  address VARCHAR(50),
                  city VARCHAR(30)
                )""".format(self.tableName)
            )
```

```
            db.commit()

    def __get_db(self):
        """
        Get a connection to the database.

        :return: Database connection.
        """
        # Get an active connection to the database if there is one.
        db = getattr(g, '_database', None)
        # Create a new one of no connection was available.
        if db is None:
            db = g._database = sqlite3.connect(current_app.config['DATABASE'])

        db.row_factory = sqlite3.Row

        return db

    def add(self, name='', email='', address='', city=''):
        """
        Add a customer to the table

        :param name: Customer name.
        :param email: Customer email.
        :param address: Customer address.
        :param city: Customer city.
        :return:
        """
        # Do the query to insert the data.
        self.query_db(
            'INSERT INTO {}(name, email, address, city) VALUES(?,?,?,?)'.format(
                self.tableName), [name, email, address, city])

    def getAll(self):
        """
        Get all customer entries.
        """
        return self.query_db('SELECT * FROM ' + self.tableName)

    def query_db(self, query, args=(), one=False):
        """
        Helper function to do a query on the database.

        :param query: The SQL command.
        :param args: The arguments to the SQL.
        :param one: Set to true to return only the fist result.
        :return:
        """
        # Default is to empty list.
        ret = []
        # Execute the query.
        db = self.__get_db()
        cur = db.cursor()
        cur.execute(query, args)
        db.commit()
```

```
        # Get all data.
        values = cur.fetchall()
        # Close connection
        cur.close()

        # If there is stuff in the result.
        if values:
            if one:
                # Return the first if thats what we were asked
                ret = values[0]
            else:
                # Return all results
                ret = values

        # Return the result
        return ret
```

6.5. app/views/add.py

```
# -*- coding: utf-8 -*-
"""
Name: Add customer view.
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0.0 (2017-04-30)
"""
from flask import render_template, url_for, redirect, request
from flask.views import MethodView

from app.models import CustomerDB


class Add(MethodView):
    '''
    View to add a customer
    '''
    # Only support GET and POST requests to get data from the form.
    methods = ['GET', 'POST']

    def get(self):
        '''
        Render the add viewn.
        '''
        # Render the template for the customer table.
        return render_template('add.html')

    def post(self):
        '''
        Handle submission of a new customer.
        '''
        name = request.form['name']
        email = request.form['email']
        address = request.form['address']
        city = request.form['city']
        CustomerDB().add(name, email, address, city)

        # Redirect to the customer table.
        return redirect(url_for('table'))
```

6.6. app/views/customertable.py

```
# -*- coding: utf-8 -*-
"""
Name: Customer table view.
Author: Martin Bo Kristensen Grønholdt.
Version: 1.0.0 (2017-04-30)
"""
from flask import render_template
from flask.views import MethodView

from app.models import CustomerDB


class CustomerTable(MethodView):
    '''
    View to show the customer data
    '''
    # Only support GET requests.
    methods = ['GET']

    def get(self):
        '''
        Render the status information.
        '''
        # Render the template for the customer table.
        return render_template('table.html', customers=CustomerDB().getAll())
```

6.7. app/templates/add.html

```
{% extends "base.html" %}
{% block title %}Add customer{% endblock %}
{% block heading %}Input customer data{% endblock %}
{% block content %}
<form id="comment_form" action="{{ url_for('add') }}" method="post">
    <div class="form-group">
        <label for="name">Name</label>
        <input type="text" class="form-control" id="name"
               placeholder="Name" name="name" required>
    </div>
    <div class="form-group">
        <label for="email">Email</label>
        <input type="email" class="form-control" id="email"
               placeholder="Email" name="email" required>
    </div>
    <div class="form-group">
        <label for="address">Address</label>
        <input type="text" class="form-control" id="address"
               placeholder="Address" name="address" required>
    </div>
    <div class="form-group">
        <label for="city">City</label>
        <input type="text" class="form-control" id="city"
               placeholder="City" name="city" required>
    </div>
    <button type="submit" class="btn btn-default">Submit</button>
</form>
```

```
{% endblock %}
```

6.8. app/templates/base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- Latest compiled and minified CSS -->
    <link rel="stylesheet"

href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
          integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
          crossorigin="anonymous">
    <title>{% block title %}{% endblock %}</title>
    <style>
        .footer-text {
            text-align: center;
        }
        {% block css %}{% endblock %}
    </style>
</head>
<body>

<div class="panel panel-primary">
    <div class="panel-heading">{% block heading %}{% endblock %}</div>
    <div class="panel-body">{% block content %}{% endblock %}</div>
    <div class="panel-footer footer-text">Customer database version 1.0.0
    <br/>
    <small>
        &copy; Copyright 2017 Martin Bo Kristensen
Gr&oslash;nholdt.</small></div>
</div>
</body>
</html>
```

6.9. app/templates/table.html

```
{% extends "base.html" %}
{% block css %}
.add_customer {
float: right;
color: white;
text-decoration: none;
}
.add_customer:hover {
float: right;
color: lightgray;
text-decoration: none;
}
{% endblock %}
{% block title %}Customer database{% endblock %}
{% block heading %}Customers <a href="{{ url_for('add') }}"
```

```
                                    class="add_customer glyphicon glyphicon-
plus"></a>{% endblock %}
{% block content %}
<table id="customer_table"
       class="table table-striped table-bordered table-condensed">
    <thead>
    <tr id="customer_table_row_header">
        <th id="customer_table_id_header">ID</th>
        <th id="customer_table_name_header">Name</th>
        <th id="customer_table_email_header">Email</th>
        <th id="customer_table_address_header">Address</th>
        <th id="customer_table_city_header">City</th>
    </tr>
    </thead>
    <tbody id="customer_table_body" style="overflow: auto;">
    {% for customer in customers %}
    <tr id="customer_table_row_{{ customer.idCust }}">
        <td>{{ customer.idCust }}</td>
        <td>{{ customer.name }}</td>
        <td>{{ customer.email }}</td>
        <td>{{ customer.address }}</td>
        <td>{{ customer.city }}</td>
    </tr>
    {% else %}
    <tr id="customer_table_row_none">
        <td colspan="5">No customers in the database</td>
    </tr>
    {% endfor %}
    </tr>
    </tbody>
</table>
{% endblock %}
```