

# Cousework2

**WEIGUANG RAN 40412989**

**Edinburgh Napier University**

**SET09103**

**Abstract:** This is a personal project so the web-app that you will build for this second coursework is completely up to you. However it is advisable to discuss your idea with the module leader in advance to ensure that it has sufficient scope to enable you to achieve a decent grade. Some ideas for suitable projects include a simple Twitter clone or a Bookmarking/URL shortening web-app. You should carefully consider the nature of the problem domain, and design a URL hierarchy and user interface that provides your users with a good experience. You should also research similar web-apps that you can use as a benchmark against which to measure the functionality of your own.

**Keywords:** Flask, web design, python

## Introduction:

### Files introduction in my app:

---**Static:** Save static files. Static files are those that do not change. In general, in my app, this includes CSS files, Javascript files and images. It can also include video files and other possible things.

---**Templates:** Templates are files that contain static data as well as placeholders for dynamic data. A template is rendered with specific data to produce a final document. Flask uses the Jinja template library to render templates.

---**Api:** save api files, and the files base on restful api script.

---**Main:** the main file is the Bookshelf application modules, it initializes them as python packages.

---**Users:** for registering and logging when people what to use the website, based on flask-wtf, flask login and flask admin.

---**Config.py:** This is the place where Flask itself puts certain configuration values and also where extensions can put their configuration values

---**Factory.py:** Creating the application object when the blueprint is imported. The aim is to:

(1) Testing. Can have instances of the application with different settings to test every case.

(2) Multiple instances. Can have multiple instances of the same application running in the same application process which can be handy.

---**Forms.py**: This file is based on flask-wtf, the form is the basic element that lets users interact with our web application. Flask alone doesn't do anything to help us handle forms, but the Flask-WTF extension lets us use the popular WTForms package in our Flask applications. This package makes defining forms and handling submissions easy.

---**Util.py**: This file is based on flask-api-utils, which aimed to help us to create APIs. Also it makes responses in appropriate formats. Another useful feature is an authentication. To sum up, it is an API example project.

---**Manage.py** : Manage the whole file, but do not need all commands in Manage.py, just need one example, and then we can call based on one example.

## Project introduction:

Name: NDI website(Napier dictionary index website)

Style: open review site

Functions: Login, register, security, personal information, database(mangodb), create label, edit label, search label, random search

Special Functions: comparing search

## Summary:

This website is a wiki-based free commenting and editing website for university. Students can evaluate the school's curriculum, teachers, teaching quality, hygiene and so on. The characteristics of this website are: freedom, openness and editability. Students can not only publish their own comments widely within the scope of the rules, but even when they find posts and content that do not meet their requirements, they can directly create tags to allow others to comment on their own tags. If you don't know how to evaluate, the navigation page of the website is also very friendly. You can provide targeted search and random search. There are also comparison functions to compare the impressions of teachers or similar courses in the hearts of students. Of course, all comments on the website are only considerable, providing students with a reference.

## Process:

### User Authentication:

Most programs require user tracking. When the user connects to the program, the user is authenticated. Through this process, the program knows its identity and

the program knows who the user is and can provide a targeted experience. And the way I used is required the user to provide a proof of identity (user's email or username) and a password. The list of packages I use is as follows:

- Flask-Login: Manages user sessions for logged in users.
- Werkzeug: Calculate the password hash value and check it.
- itsdangerous: Generate and check the cryptographic security token.
- Flask-Mail: Sends emails related to authentication.
- Flask-Bootstrap: HTML template.
- Flask-WTF: Web Form.

### **The security of password:**

When designing a Web application, we tend to overestimate the security of user information in the database. If an attacker invades the server to get the database, the user's security is at risk, and the risk is much larger than we think. As we all know, most users use the same password in different websites, so even if no sensitive information is saved, the attacker can access the account of the user on other websites after obtaining the password stored in the database. So I use the following things to ensure password security:

Implement password hashing with Werkzeug:

The security module in Werkzeug makes it easy to calculate the cryptographic hash value. The implementation of this feature requires only two functions, which are used in the Registered User and Verify User phases.

**\*\*\*generate\_password\_hash(password,method=pbkdf2:sha1, salt\_length=8):**

This function takes the original password as input and outputs the hash value of the password as a string. The output value can be saved in the user database. The default values for method and salt\_length will satisfy most requirements.

**\*\*\*check\_password\_hash(hash, password):** The parameters of this function are the password hash value retrieved from the database and the password entered by the user. A return value of True indicates that the password is correct.

**PS:** The function that computes the password hash value is implemented by a write-only property called password. When setting the value of this property, the assignment method will call the generate\_password\_hash() function provided by Werkzeug and assign the result to the password\_hash field. If you try to read the value of the password attribute, an error is returned for obvious reasons, because the hash value cannot be restored to the original password.

### **Use Flask-Login to authenticate users:**

After the user logs in to the program, their authentication status is recorded so that they can be remembered when browsing different pages. Flask-Login is a very useful small extension designed to manage the authentication status in a user authentication system without relying on a specific authentication mechanism.

```
<div class="header">
  Napier Dictionary Index login
</div>
<div class="content">
  <p>Do not have the account?<a href="{{ url_for('users.register') }}">Click </a>register
  <div class="ui ignored error message" id="error" style="..."></div>
  <div class="ui ignored positive message" id="success" style="..."></div>
  <form class="ui form" id="lg_form">
    {{ lg_form.csrf_token }}
    <div class="field">
      <label>{{ lg_form.email.label() }}</label>
      {{ lg_form.email() }}
    </div>
    <div class="field">
      <label>{{ lg_form.password.label() }}</label>
      {{ lg_form.password() }}
    </div>
    <button type="submit" class="fluid ui teal button" id="sign_in">login</button>
  </form>
  <br>
</div>
</div>
```

div.ui.modal > div.content > form#lg\_form.ui.form > button#sign\_in.fluid.ui.teal.button

## Register new use:

If a new user wants to be a member of the program, they must register in the program so that the program can recognize and log in to the user. A link is displayed in the login page of the program, bringing the user to the registration page, allowing the user to enter an email address, username and password.

And for security reasons, the password is entered twice. At this point, I decide verify that the values in the two password fields are consistent, this verification can be implemented using another validation function provided by WTForms, namely `EqualTo`. This validation function is attached to one of the two password fields, and the other field is passed as a parameter.

What's more, for certain types of programs, it is necessary to confirm that the information provided by the user at the time of registration is correct. A common requirement is to be able to contact the user via the provided email address. To verify the email address, the program will immediately send a confirmation email when the user signs up. The new account is first marked as pending, and the user can follow the instructions in the email to prove that they can be contacted. During the account confirmation process, the user is often asked to click on a special URL link that contains a confirmation token. So I use `itsdangerous` package:

```

def add_edit(self):
    mongo.db['users'].update(
        {'_id': bson_obj_id(self.id)},
        {'$inc': {'edit_count': 1}}
    )

    @staticmethod
    def gen_passwd_hash(password):
        return generate_password_hash(password)

    @staticmethod
    def verify_passwd(passwd_hash, passwd):
        return check_password_hash(passwd_hash, passwd)

    def gen_auth_token(self, expiration):
        s = Serializer(current_app.config['SECRET_KEY'], expires_in=expiration)
        return s.dumps(bson_to_json({'id': self.id}))

    @staticmethod
    def verify_auth_token(token):

```

## User Information:

In order to make the user's profile page more attractive, you can add some additional information about the user, so I decided to expand the User model.

```

        if user['avatar']:
            fs.delete(bson_obj_id(user['avatar']))
            data['avatar'] = avatar_id
        else:
            flash('图片格式不支持', 'red')

        User.update_user(user['_id'], data)

        return redirect(url_for('.profile'))
    else:
        flash('资料修改失败', 'red')
    return render_template('profile_edit.html', user=user, form=form, title='编辑资料')

@sers.route("/logout")
@login_required
def logout():
    logout_user()
    return redirect(url_for('main.index'))

```

## Database:

To post a comment on the support site, I need to create a new database model. Finally, by comparison, I decided to use the MANGODB database. The reason why I use this database is that mongodb can be allowed in Flask, compare with mysql, mongodb is easier to implement on small sites, and data changes are better.

Mongodb is one of the Non-relational database. The tables in a relational database stores some structured data. The fields of each record have the same composition. Even if not all the fields are needed for each record, the database allocates all the fields for each data. Non-relational databases are stored as

key-values. Their structure is not fixed. Each record can have different keys. Each record can add some key-value pairs as needed. Limited to fixed structures, can reduce some time and space overhead

```
from app import app
from app.extensions import mongo
from app.models import Item, User
from flask.ext.script import Manager, Shell

manager = Manager(app)

def make_shell_context():
    return dict(app=app, db=mongo.db, Item=Item, User=User)
manager.add_command("shell", Shell(make_context=make_shell_context))

if __name__ == '__main__':
    manager.run()
```

## API:

Introduction to REST:

Resources are the core concept of REST architecture. In the REST architecture, resources are things you want to focus on in the program. For example, in a blogging program, users, blog posts, and comments are all resources. Each resource is represented by a unique URL. The format or content of the URL does not matter, as long as the URL of the resource represents only a single resource.

Eg:app/api/auth.py

```
from app.extensions import mongo
from app.models import User
from flask import g, jsonify
from flask.ext.httpauth import HTTPBasicAuth
from . import api
auth = HTTPBasicAuth()
@api.verify_password
def verify_password(email_or_token, password):
    user = User.verify_auth_token(email_or_token)
    if user is None:
        user = mongo.db.users.find_one({'email': email_or_token})
    if not user:
        return False
    else:
        if not User.verify_passwd(user['password'], password):
            return False
```

```

        g.current_user = User(user.pop("_id"), extras=user)
    return True
@api.route('/token')
@auth.login_required
def get_auth_token():
    token = g.current_user.gen_auth_token(600)
    return jsonify({'token': token.decode('ascii'), 'duration': 600})
@api.route('/resource')
@auth.login_required
def get_resource():
    return jsonify({'data': 'Hello, %s!' % g.current_user.username})

```

## How to use the website:

### Putty:

- 1 open putty and login
- 2 Clone github url: [https://github.com/oliverran/set09103\\_coursework2\\_4041](https://github.com/oliverran/set09103_coursework2_4041)
- 2989-WEIGUANG-RAN.git
- 3 download packages: pip install -r requirement.txt
- 4 Enter **app/config.py** on the command line
- 5 Enter python manage.py run

### Pycharm:

How to use:

## download packages: pip install -r requirement.txt

## Configuration file: app/config.py or

## Run: gunicorn wsgi: **app -c gunicorn.conf** Or **python manage.py**

## Weak and Thought:

1. UI design: I do not pay much attention to UI design, so it is not very friendly to people and need some update according to people's feedback. Eg: like the main page, the visual comparison is monotonous and the search box is too small.

2. Comments and wiki function: Although the types of comments are comprehensive and there are many types, they are not clear enough. So if I want to update in the next step, I need to start with the user experience.

3. The comments on our website are limited to public comments. People can find the relevant content directly through the search, but this is not enough privacy and personalization, so next time I am going to set up a personal homepage, everyone can freely post content on the personal homepage. And these content can be set to be visible to the public.

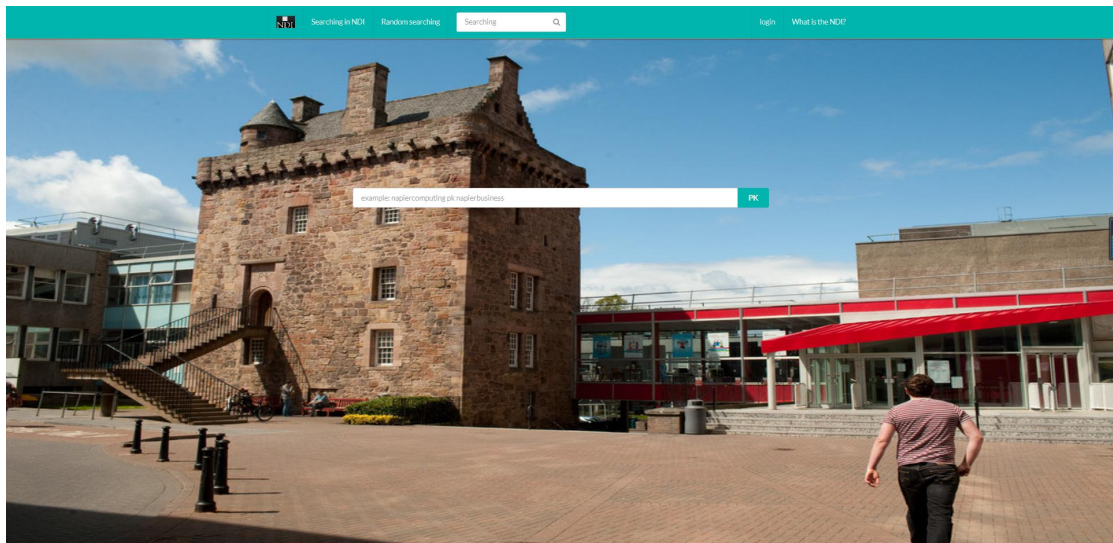
4. Fuzzy search: This is an important thing which I should be solved in the future.

From now, my website can not searching precise things, like if you want to search simon wells, you must input 'simon wells', all letters, spaces and capitalization can't be wrong, but this is obviously unreasonable, because it's impossible for everyone's searching is totally true every time.

But fuzzy search is quite difficult, I tried to achieve it and I still have no idea so far, but it will be very useful if my website has this function.

## Appendix:

Index:



Register:

A screenshot of the NDI registration form. The form is titled 'NDI register to NDI' and includes four input fields: 'mail', 'user(length between 2 to 18!)', 'password(length between 2 to 18!)', and 'repeat password'. A teal 'register now' button is positioned below the fields. At the bottom, there is a link that says 'Already have account?login'.

Login:



Napier Dictionary Index login

Do not have the account?[Click](#) register

mail

password

login

Create:


BASE


name


catelogue


Select type


Profile:




 rwg19970614

 无

 [oliverranweiguang@gmail.com](mailto:oliverranweiguang@gmail.com)

 无

 2018-11-11

Edit your data

Introduction

无

Contribution

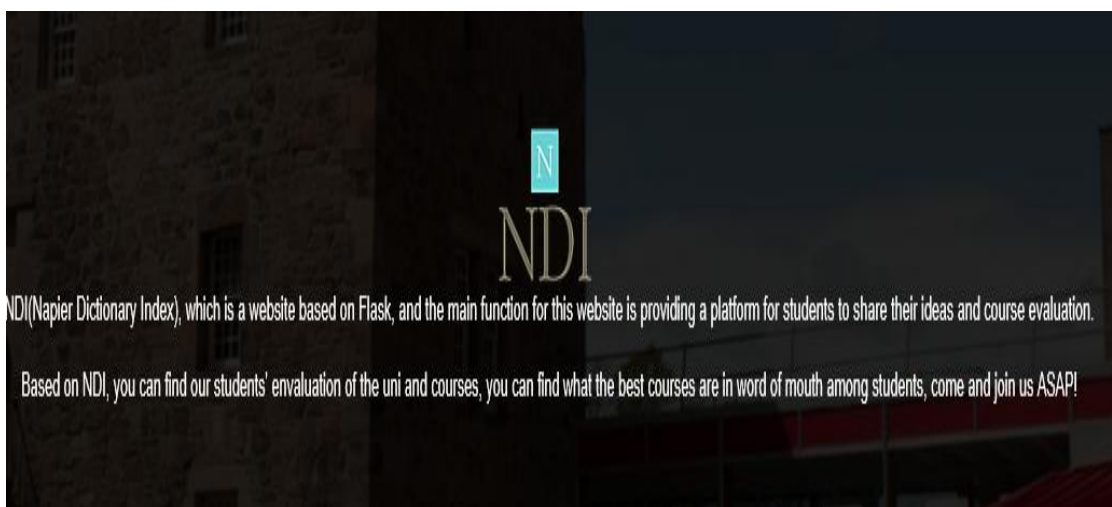
create the entry  
无

edit your entry  
无

PK:

Comparison items	Simon	TAOXIN PENG
Module	Computing	Data analytic
lecture	90 lecture	?
grade	lecture	?

About:



Script:



redis==2.10.3

requests==2.8.1requests-oauthlib==0.5.0

Werkzeug==0.10.4

WTForms==2.0.2