SetCloud

Louis Boursier
40404293@napier.ac.uk
Edinburgh Napier University - SET09103

Abstract

This document is a report for the second coursework in the advanced web technologies module[1]. For this coursework, we had to build a web application prototype. The objective is to demonstrate our understanding of server-side web development, mastery of the Python Flask micro-framework and the role of HTTP and related protocols in the design of robust and scalable web application. We should achieve this by completing a personal project in which we design, implement, and evaluate a web application on a topic of our choice.

Keywords – Louis, Boursier, Flask, Python, Web, application, set, cloud, SetCloud

1 Introduction

I chose to build a website which provides a file management system based on the mathematical set. The idea is to create a new way to classify, organize, and retrieve files. This kind of file management has not for objective to replace the current used tree based hierarchy file system, but to give the user with a different way of managing his files. The idea of using mathematical set and not nested folders also extends to a right and privilege administration over those files. I will talk more about this later in the enhancement section

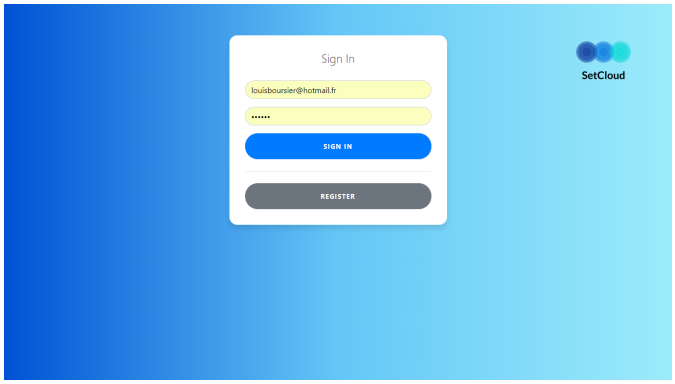

Figure 1: **Website home page** - The user should always go through the identification page to connect or create an account in order to use the application

The home page is the entry point of the application. This ensure that the user is always identified and verified. From this page, the user can either create a new account with a valid an non existing email address, or login thanks to an already existing account. If the email chosen to create an account is already in use, the process is aborted and the user is notified. When trying to log in, the same thing append if the credentials are invalids.

Figure 2: **Main page** - This page displayed the upload input, followed by the search input and the set visualizer

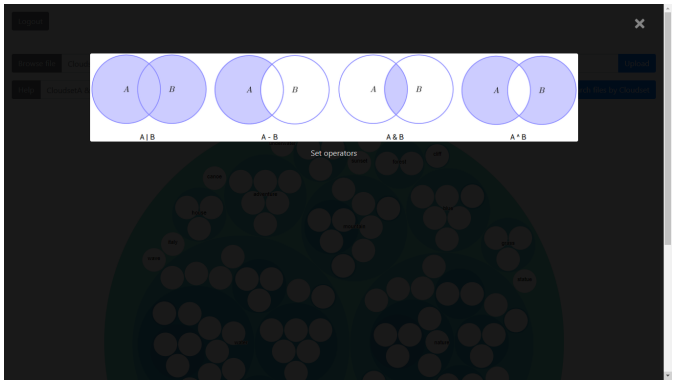

Figure 3: **Help for the set operations** - This pop up is launched when the user clicked on the help button. It explains which characters correspond to which set operations

On the main menu page, the first input is for the file uploading. The user choose one file to upload (it can be any kind of file, but for a security safety, I only allowed the most basics files like images and PDFs). After having chosen the file, the user should specify the SetCloud (the set's name) corresponding to this file. The more sets the user can specifies, the better the search and classification will perform. For each file, the default set is always implicitly applied. This ensure that there is no file without SetCloud, and that the user can get his hands on all his files. It also makes the set operations work better. Once the file uploaded, the file will transparently be saved in

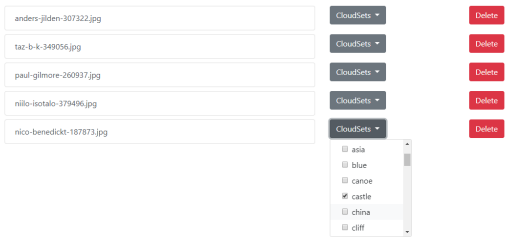

Figure 4: **Files visualization after using the search input** - Here are shown the files for which their set correspond to the input search command. The user can edit the set for each file, and delete them.

the user's google drive account (for which he has been prompted to connect before). All the SetCloud files will be in a same folder in his account, for which he should not touch or even know the existence. As soon as a file has been uploaded, the visualization at the bottom of the page is updated to match the current state of the files and their SetCloud.

The second search input is for searching file using set operations. If no set is specify by the user, only the default set will be applied, which will have for effect to print all the user's files. Otherwise, the user could use as many set and set operations has he wants. For the moment, the parenthesis are not supported. Hence, the logical order is from left to right.

$$SetA - SetB - SetC|SetD \quad (1)$$

The files page is displayed after a search. It is displayed there all the files for which their sets correspond to the previous command. From this page, the user can add or remove a SetCloud for a file, and delete a file. All those operations are done thanks to AJAX, meaning that the user does not have to reload the page. A notifiication pop up to inform the user of the success of the operation.

2 Design

3 Enhancements

In this part, I will elaborate on what can be improved or do to improve the existing project.

3.1 Add a right and privilege system

Using the same mathematical set concept, I could build a right system leveraging the sets. Like for the file search, this could allow more efficient and powerful way of setting files rights. I can think of different subgroups levels, from the basic user, to the administrator and the root user. The root having the administrator for subset, and the administrator having the user for subset. Rules can also by applied conerning files for the groups thanks to the already existing file search.

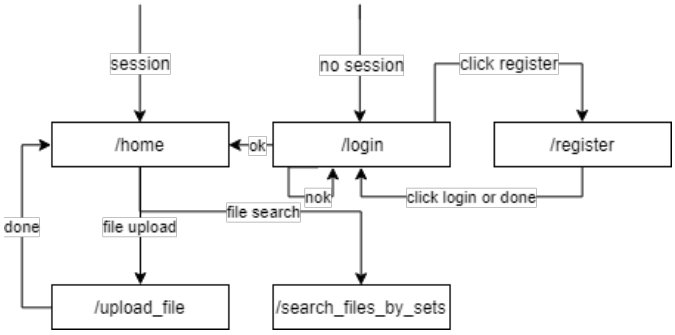

Figure 5: **URL Hierarchy**

3.2 SetCloud proposal for images

Using AI (like the Amazon Rekognition API), I could suggest SetCloud name for images based on the image content.

3.3 Extend the SetCloud visualizer

Like the windows file explorer, I want my visualizer to print all the files when the user's click targeted a leaf circle. This would print the files for the corresponding SetCloud with an implicit set operation command, without the user having to type it by himself.

4 Critical Evaluation

I have bulit a simple web application prototype, using Flask micro-framework and the Python language. I have also used the HTML and CSS language associated with Flask to generate HTML pages. I have dynamize the website using Javascript and the JQuery framework. And I have used Sqlite and the SQL language to handle the database.

4.1 Object Oriented

Even though the Flask Application object could be challenging to use in an OO paradigm, I have build my website in this way. This ensure a scalable application, and the fact that each part is separated in a specific file with its class make the whole project easier to understand. It is also useful for the unit testing to build the project that way.

4.2 MVC

I leveraged the OO paradigm to build my application based on the Model View Controller design pattern. Like for the OO, it allows the code to be clearer, and it simplifies the future implementation for the code.

4.3 Security

For this project, I have used Jinja built-in protection against XSS attacks. I also used prepared statement to avoid SQL injection. And I have used BCrypt to manage the passwords hashes. I ensure the identity and the rights of the user thanks to Flask Session.

Listing 1: Query with SQL parameters

```
1 def set_exists(self, setName, userEmail):
2         self.conn = sqlite3.connect(self.dbFileLocation)
3         res = self.conn.cursor().execute('SELECT cloudset.name ↩
          FROM cloudset INNER JOIN user ON user.id = cloudset.↩
```

```
        userId where user.email = ? AND cloudset.name = ?', (↩
        userEmail,setName,)).fetchall()
4       return res
```

4.4 BCrypt and Flask Session

Listing 2: Login with BCrypt and Session

```
1 def login(self):
2       if request.method == 'POST':
3           db = Database("var/sqlite3.db", True)
4           # no htmlentities() or equivalent because flask uses its ↩
        own xss protection while using render_template()
5           email = request.form.get('inputEmail')
6           inputPassword = request.form.get('inputPassword')
7           res = db.get_user_password(email)
8           if not res or not self.bcrypt.check_password_hash(res↩
        [0][0], inputPassword):
9               flash('Wrong credentials.')
10          else:
11              session["user"] = email
12      return self.indexPage()
```

4.5 Flask in OO

Listing 3: Login with BCrypt and Session

```
1 def main():
2    app = Flask(__name__)
3    app.secret_key = os.urandom(24)
4    app.config['SESSION_TYPE'] = 'filesystem'
5    Session(app)
6    app.debug = True
7    controller = Controller(app)
8
9    app.add_url_rule('/', 'indexPage', lambda: controller.↩
        indexPage())
10   app.add_url_rule('/register', 'registerPage', lambda: controller↩
        .registerPage())
11   app.add_url_rule('/home', 'homePage', lambda: controller.↩
        homePage())
12   app.add_url_rule('/logout', 'logout', lambda: controller.logout↩
        (), methods=['POST'])
13   app.register_error_handler(404, lambda x: controller.↩
        page_not_found())
14   app.run(host=HOST, port=PORT, debug=DEBUG, ↩
        use_reloader=False)
15
16 class Controller:
17   def __init__(self, app):
18       self.app = app
19       self.bcrypt = Bcrypt(self.app)
```

5 Personal Evaluation

During this project, I mainly learned about the Flask micro-framework. As I have already built a website application with the Python Bottle micro-framework [2], the learning was easier thanks to the fact the they share common traits. In the same way, I have also learn to build unit test for Flask. The main difficulty was to think about the most efficient and coherent URL hierarchy according to our data and the website I wanted to build. To do so, I have put myself in the place of the user, and have think about what they would need and how. Otherwise, the project was quite a déjà vu for the reason I told above.

References

[1] S. Wells, "Teaching SET09103," 2018.

[2] M. Hellkamp., "Bottle: Python Web Framework," 2018.