Edinburgh Napier UNIVERSITY

Coursework Report

Rafal Ozog
40217610@live.napier.ac.uk
Edinburgh Napier University - Advanced Web Technologies (SET09103)

Keywords – rafal, ozog, report, web, webapp, coursework, antiauction, python, jinja2, html, css, templates, webdevelopment, web-development, json, api, url, url hierarchy

1 AntiAuction - "Buy, sell, compete for the best price!"

Since the Internet became a part of our social life, people have quickly realized that it can be a powerful tool, able to provide services like online trading and shopping. We can buy and sell - just in one click - cars, clothes, services, media, properties, food and probably anything we can imagine. That was the fundamental idea for AntiAuction web service creation. For the first glance it seems to be similar to other online shopping platforms, like Ebay - however, the idea is a little bit different - it is going to be a novelty. Let me explain that riddle from the beginning...

2 Front-end Design

The application provides a number of functions that all together are composed into one, compact and efficient online trading platform. I will start my explanation from brief overview of the most significant features and functions, going from the outside (Front-end) to the core (Back-end). In this moment I will only mention that in technical meaning this website is **a collection of offers** (items to be sold).

2.1 Navigation - where can I go?

The spine of the website is a navigation toolbar, placed on top of the window, just under the logo. That is the main mean of transport throughout the application, allowing users going to "fresh offers" (main page), internal search engine, selling menu, account interface and logout gate with just one click.

2.2 Fresh offers - the main site

The easiest way to explain something is to show an example of how does it work. Thinking about this, I have decided to place the newest offers on the main page, which will allow people to realize what is that website about, just at one glance.

There are 10 newest offers (retrieved from the 'json' file) displayed on a front page. When there are more than

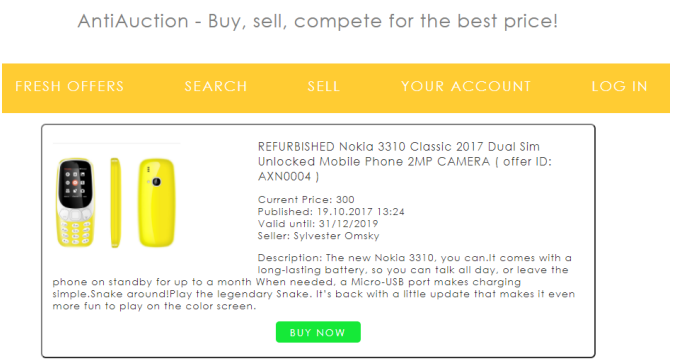

Figure 1: **Fresh offers** - the main page layout

10 offers in our database, then a 'next page' button appears.

2.3 Search for what you're looking for

The second sub-page contains the internal search engine. User can specify the phrase which will be filtered out of the titles and descriptions of all current offers. Results are presented in the same form as they are displayed on the main page. As the searching results contain 'buy' button, the logged-in user can buy the found item immediately.

2.4 Are you logged in?

As trading is strictly related to money transfers I've decided to implement logging mechanism. Main reason of this solution was to make transactions safe and more private.

There are number of features, which are restricted for users that are not logged in. The anonymous user will not be able to publish his own offer, to buy anything and of course to manage his account. I will explain the logic of this mechanism later on (in the back-end section).

2.5 Registration

The application design was planned to be simple and readable. To create an account the user has to enter his login, password and e-mail address (basic credentials indicated by a star). In the same form, he can also provide his personal details and correct address, which will be necessary to proceed any transaction. Once the client will click on the button, the account will be created and saved into the database (which is a 'json' file). He will become logged in automatically. Since that moment, the

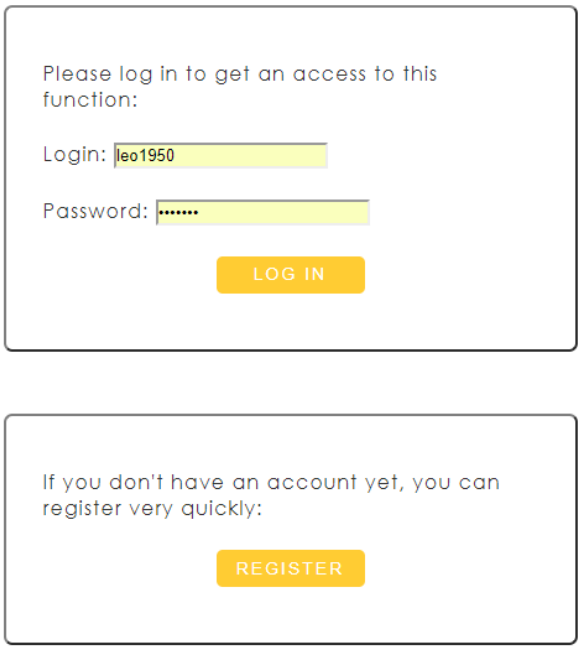

Figure 2: **Login page**

user will be authorized to use the whole functionality of the web-app.

2.6 You always can log out

Sometimes user wants to exit the room. We can imagine a situation, where someone is using a public computer or a few people are using the same laptop. Because of these reasons I realized how important it is to place the 'log out' button into a visible and easily-accessible place - the navigation toolbar.

2.7 Do you want to sell something?

The selling function is available only for registered and logged-in users. If an anonymous person will try to access it, then will be redirected to the 'login/registration' page. To sell an item user can choose an offer title, describe briefly what is that thing, upload a photo, select an expiry date and set a minimal and maximal price. The maximal price is the proper price displayed publicly. The minimal one is related to a feature that will be developed in the future. My idea is to create a service, in which the offer is published with the starting price, which is decreasing with the time, until the minimal value is reached. That is exactly the **unerlying concept** that is going to distinguish my website among other trading platforms. It is going to work as an inverted auction as users are hunting for the best price before an offer will be expired or taken. If the form is filled in correctly, a proper statement confirms that fact after the button click.

2.8 Check your account

Another function available on the navigation toolbar is an account interface. In this section user can check his personal/contact data and see the summary of all offers published by him. There are also few options which are not accessible in this moment, but will be developed soon. These functions are user data editing, displaying bought items and account deleting.

3 Back-end Design

3.1 Used technologies - overview

The whole application is based on a Python functions and organised **URL hierarchy**, which allow to react dynamically on user's requests. Web pages are prepared in HTML, organized in correct templates supplied in some jinja2 code, which provides communication and data transfer between Python and HTML code. Following the concept of structure/layout separation I have provided (by myself) some CSS stylesheet which describes how the HTML elements should look like. Data is organized and stored in json file.

3.2 URL hierarchy

The application architecture has been based on a semantically organized URL hierarchy. Features are grouped into sets 'user' and 'offers', depending if they are more logically related to the first or the second group. The main page has been excluded out of this division as the main reference point of the whole URL structure. Each of the URL paths corresponds to one particular application function. Some of them are displaying HTML templates, some receive and proceed data (using http 'POST' request) and some other provides a particular part of the Python code functionality. This structure form an **API** of my web application, which allows to access different sub-pages and their functions using the particular **URL paths**.

```
'/ '
'/ page / <number>'

'/ offers / sell'
'/ offers / sell / published'
'/ offers / search'
'/ offers / buy / <id>'

'/ user / login'
'/ user / logout'
'/ user / register'
'/ user / account'
```

Figure 3: **URL hierarchy** - an underlying application structure

The connections between different sub-pages and func-

tions are provided both through **'redirect'** function inside the Python code and through simple 'a href' HTML tag within the templates. This ordered structure can be easily developed and extended.

3.3 Python code division

I have organized my code in a few separated 'py' files to make it more readable and easier to modify. The content of each file correcponds to one particular branch of the application functionality - there are files **'users'**, **'data'** (related to offers data), **'searching'** and **'index'** (the main program file with the URL hierarchy).

3.4 Data storage

The data is organized in a few 'json' files. Users data has been isolated from offers data to make possible exporting particular parts of that 'database' separately. The access to the 'json' files is performed by special Python functions I have created ('get number', 'read offer', 'add offer' etc.).

3.5 Search engine

The code of this function is located in separate 'py' file. The mechanism takes the phrase entered by user and goes through the list of all offers, trying to find that phrase within a title or a description. The code consists of 2 similar functions - one of them is looking for user's phrase, second one is matching proper offers (published by him) to be displayed at his profile (account) page.

3.6 Password encryption

I have implemented a number of security features to provide the highest level of the user's safety I could. First simple, but important thing is that password credentials are replaced by stars, not to allow other people to read the password from our screen. Secondly, the password is salted and hashed by **'bcrypt'** functions. When a user is registered, the 'add user' function provides the encryption and sends a password **hash** to the 'json' file. When someone tries to log in, the decrypting function ('login' function) compares the hash of user's credential with the stored password hash. If both are identical, the user gains the access.

Note: because of the password encryption, there is a library **'bcrypt'** needed to be installed and imported to allow the application work correctly. This is **the only external** (installation requiring) **library** used by the program.

3.7 Session storage

The application is using **Session storage**. That is an example of functionality that exceeds the strict user(request) - server(response) model, because some part of data is stored within the user's browser, which became involved in the interaction. The session has been used to store two informations - a user status (logged in or not) and (if logged) a user's login, what allows the browser to display proper account details and offers. When the user decides to log out, the session is cleared.

3.8 Login mechanism

I have already mentioned that some functions are restricted for users that aren't logged in. Now I would like to explain more deeply how it works. The core of this solution is a special function decorating functions that should require **authorization** (the function 'check if logged' in my code). The only task of this mechanism is to check the user's status in the browser session. If the status is not confirmed as 'logged in', then the user is redirected to the login/register page.

3.9 Arguments passed within the URL

I have concluded that in some particular situations it will be an appropriate to send some data through the URL. There are two functions in my code using this solution - a 'buy' function, which passes an ID of bought item and a mechanism redirecting user to the next page of fresh offers, which passes a page number.

3.10 A config file

Although the config file contains only two pieces of information that the application uses int this moment (ip address and a proper port number), I have decided to create and use it. The reason was that it may be useful for a future development, which may involve more data stored in the config file.

4 Future perspectives

I think we can compare web-development to a kind of art. Someone said 'the painting process is never finished, only interrupted'. We could say the same about application. Even the most perfect piece of art could be improved in some way. In this section I would like to present some features and functions I would like to implement in my application, in the future.

4.1 Price decreasing

The most important function I would like to add to my website would be the price decreasing. That is the idea, which I have explained more deeply in a section 2.7. The fundamentals of this function have been already implemented. The offers contain starting (maximal) and ending (minimal) price - and also the expiry date, specified by the seller. The application needs a piece of Python code that will be calculating the correct price for each particular moment and displaying it correctly on each single offer.

4.2 Bought items

Another important feature that hasn't been implemented yet is an assignment of bought items to proper users. In this moment, when the 'buy' button is pressed, the offer is deleted from 'offers' database and moved to 'archive'. However there is no provided information about the new owner. Because of that reason these offers can't be displayed on the 'account' page as 'purchased things'. My idea to implement this feature will be the same as for assigning published offers (that is working properly). I will

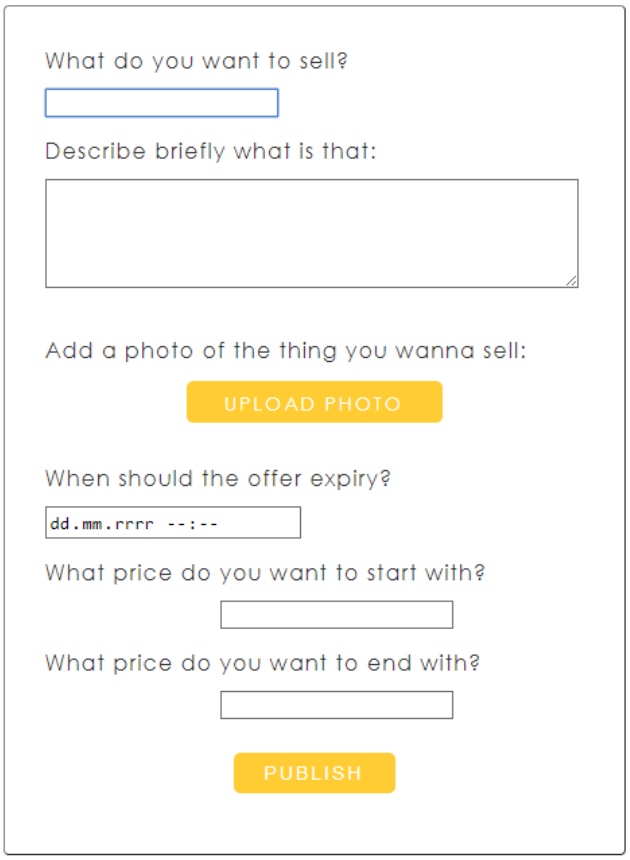

Figure 4: **Selling page** - current version

create a new variable, which will contain a user's login taken out of the session. The variable will be passed to the 'archive' database ('json' file) with other informations related to this offer (when the transaction is confirmed). This solution will allow the application to identify an owner of each offer.

4.3 Personal/contact data editing

In this moment there is an 'edit' button at the 'account' page, but it doesn't work yet. This feature will be implemented in a very simple way. I will use the function 'find user', which receives user's login (taken from the session) and returns a number of a memory cell, which is occupied by this user in the 'users' database. Later I will call another function 'read user' to retrieve that object from 'json' file to my Python code. It will give me an access to change that data. The user interface will be provided by a proper HTML template (with a proper form).

4.4 User deletion

Nowadays the user account can be removed only through 'json' file editing. In the next version I would like to make the button 'delete', which will be redirecting to a page displaying some warning communicate. If the user will confirm that choice, his account will be removed from a database 'users' using the same (or similar) function to this, which is removing offers. It is an independent function that retrieves the data from 'json' file into a Python dictionary, uses 'pop' function to remove a particular record and writes that updated data into the 'json' file again.

5 Critical Evaluation

5.1 What is done well

The first thing prepared well in my personal opinion is the **internal URL structure**. The URL paths are grouped and ordered (to read about the details, see section 3.2). I was trying to organise it logically consistently to provide a readable and easy way to use **API**, which could be extended in the future.

Another thing that works well is a **user interface**, providing a proper access both for logged-in and not logged-in users. I think that the login mechanism and user functionality have been written clearly in Python (user-related functions has been separated into an independent 'py' file), what allowed me to modify them easily and ensure that it doesn't contain any semantic errors. I have spent some time on the user interface project. I was trying to make it simple and readable, but also containing all features necessary for searching, buying and selling items. One of this thinking results is a navigation toolbar.

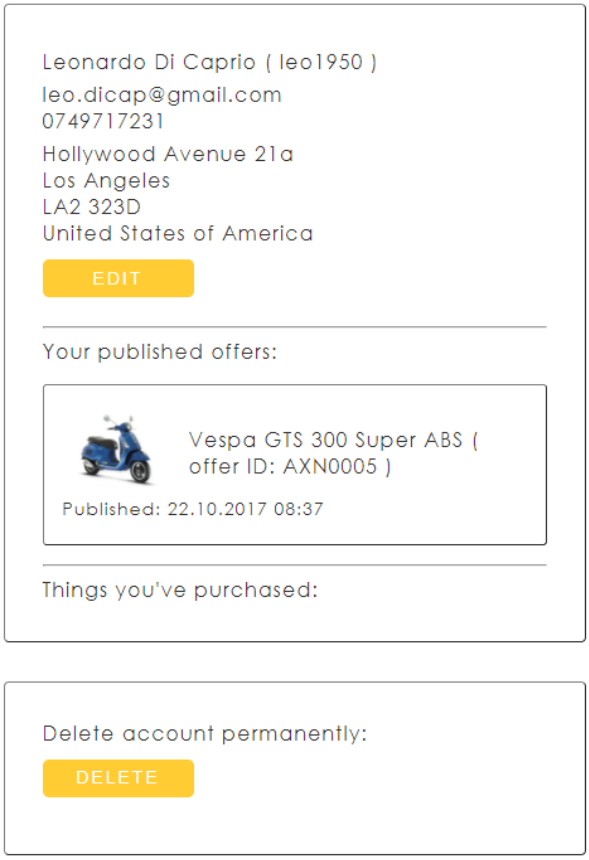

Figure 5: **Account page**

I think that the **data storage** is also well-performed feature. The functions related to data handling are located in separate 'py' file. They are accessing data stored in **'json' files**, reading their content and importing to

Python. There is also a function responsible for adding new records into the database (into 'json').

Password encryption is another example of function which I think is prepared in a proper way. There is an encrypting function, that salts and hashes the password and decrypting function. Database stores only password hashes to provide some security level (I have explained it in the section 3.5).

5.2 What could be improved

In this project I have decided to create my own CSS stylesheet instead of using Bootstrap. I was trying to prepare a friendly and readable layout, which will be also original enough to remember my page for a little bit longer than just few seconds during the Internet surfing. However, the effect is a CSS stylesheet that could be improved. It works just like I wanted, but the code could be simplified and aggregated into a smaller number of classes.

Another thing that I didn't do in a proper way was not following the HTML5 rules and mixing the modern HTML5 semantic tags with my own divs. In the next version I would like to create a layout based entirely on these tags, what would make my HTML and CSS code much simpler and easier to update.

6 Personal Evaluation - what I have learned

6.1 GitHub version control

Probably the most trivial but also the most useful programming skill I have improved working on this project is **GitHub** using. Until this moment I had already known what Git is, but I felt confused about it and basically I didn't use it. During this coursework preparation I was using Git as an every-day tool, storing my files in an online repository. I have gained confidence about working with it and also made some **'add-commit-push'** habit, which will be useful in my future programming projects.

6.2 Web app development

Before I started this module I didn't have any experience and proper knowledge about web app development. Now I know which technologies could be useful in this work (Python, jinja2, html templates) and how to use them in practice. Now I am able to combine them together and distinguish their syntax rules easily.

6.3 API, URL structure

The main concept which was totally unknown for me before I started this module, is a URL web application structure and the idea of API as an application interface. It gave me an important insight into a real web development and I basically understood how the web applications are really created.

6.4 Decorator issue

One of the troubles I had to overcome was how to deal with a **decorator**, what I needed to implement 'login' function that will be **wrapping up** other functions. I had already known this Design Pattern, but I hadn't met its Python form before. I have spent around two days learning about how the decorator should exactly look like, but I think it was a valuable experience. I could refresh my knowledge about this pattern, got to know how is it structured in Python and how to use it in practice. Finally I have created that function in a correct decorator form and it is working properly.

Listing 1: Decorator scheme I have used

```
1
2   @app.route("/url_accessible_only_for_logged_in_users")
3   @check_if_logged
4   def function_of_the_restricted_site():
5
6       (...)
7
8       return render_template('restricted_site.html')
9
```

6.5 Json

This project was for me an opportunity to learn and practise **'json' format using**. I haven't been working with this before, but during this work I found out that it is a really useful tool, which can be used for many purposes. That's a totally new thing I could add to my skillset.

6.6 Encryption

Another piece of knowledge I got during this project is related to **enryption**. I didn't have any idea how is it performed in a real applications, before I started this coursework. I learned about encryption libraries, hashing, password storage and different techniques making our app more safety.

7 My feeling about this project

At the end I would like to express briefly my feelings and thoughts about this coursework. It is always hard to evaluate our own work. I can't assess if is this project really well-done or not. However, I would like to say that I am personally satisfied with my work and a skill progress. I could learn and practice a lot, I am pretty sure that these values will not disappear quickly, but will be paid of in the future.

References

https://www.youtube.com/user/MiroslawZelent/videos

The Miroslaw Zelent YouTube - I have been learning HTML5 and CSS3 using online courses from this channel.