**Our site**

This is a site for road transport system, includes 3 user types, 1, public users 2, road safey officers and 3, staff

*Public functions*: 1, sale car 2, apply license 3, renew license 4, register car

*Functions for staff*: 1, approve/reject license application 2, approve/reject vehicle register 3, approve payment 4, revoke license

*Pages for register and login*: 1, login page 2, register page

**Register page functions**: we have a html for register with four forms that can be filled, username, password, confirm password, email. Each button is made like this:

<form action="/function" method="post">

name: <input name="button" type="type" />

They process the data and call the function of register using the button above, using @post, the function is as below:

@post('/register')

def do\_register():

when called, the function will use request.forms.get :

something = request.forms.get('something')

to get all the input values from the inputs, then inside do register check register is called using the data got from above

err\_str, register = check\_register(username, password, confirmPassword, email, memberType)e

, check register() first check if password == confirm password using check password, then it checks if a filename containing username already exist, if not it will save a file using its usernames file name and is a txt file which contains all the info, the encrypted info will be saved into the dictionary sorted by username.

**Login:** each time user login, we get data from the webpage exactly like register, and we check password. And read file from data using request, request will call a function from backend using post. Which check password and authentication from user dictionary and set currentuser . And if the user exists we will split the information from its user file so we can display it on the login page. Thus the login page will change using load\_and\_render to display the details of user details.

Saving and reading important user info: we generally save information like user password inside a dictionary called user, where we save encrypted password at user[username][0] and authentication code at [3]

**Main:** this is the page where it displays functions based on the user. Now the main page will call the function main(), which first check the currentuser using the getCurrentUser function from backend using post, which will read the currentuser data we saved from login, and check the current user rule, the according page is displayed using a if function which checks member type which is the third thus [2] after split() in the file which is found by first check currentusername and find its according userfile. Then use load\_and\_render to display the page.

Currentuser: the data of this is also saved in the dictionary, using user[currentUser], this value is cleaned upon log out

***General Page Applications***:

**public**: On the main page of public user, four functionalities are provided: sale car, register car, apply license and renew license. Sale car is associated with function sale\_car and do\_sale of the fronted and check\_sale function in the backend. The frontend post requests to backend, then backend check whether the car is destroyed and the owner of the car. If the car is not destroyed and the currentUser is the car owner, the application would be sent to staff and wait to be approved, otherwise the application is failed. The apply license function allow user to apply license. If the user already has a license, the check\_license function would return false and not send application to the staff. If the user do not has license, the request would be sent to the staff and wait for process. After the application is sent, When renew the license, the user send a request of renew license by method requests.post("{target}/api/check\_renew\_license".format(target=backend\_str)).text. Then the backend would check the associated information with the license of this user and give response. The last functionalities of the main-user page if register car, the associated function is do\_CarForm(). do\_CarForm() function requests reponse backend by method requests.post() and get response of if the car is destroyed. If the car is destroyed, then the register is not approved. Otherwise the staff would check whether the register should be approved or not and give response.

**Staff:** For staff account, it has four functions: approve/reject license application, approve/reject vehicle register, approve payment and revoke license. First of all, the approve/reject license application and approve/reject vehicle register work very similar. When users apply application, it stores it data. Staffs are able to inspect the applications, and give the result. Users are able to check the state of application in profile page. On the other hand, when users get fine, user can pay the fine online and staff will approve it. Users can only pay off all the fine to renew their license and new vehicle register. Finally, when the users break the traffic regulations too many times or involve some serious accident, staffs are able to revoke licenses. In some extreme cases staffs are able to stop some specific user’s application.

In most cases a file is saved in the directory named after the username with the application as its file type, for example, superman. ApplyLicense . Now when the staff load their approve or reject license page, the page will display all the user info with a file like this found, in this case superman, and if approved or rejected, either delete or change file type to License.

**Methods of writing files:** our site is now using backend which puts all the write functions inside a backend file, and is called from the frontend, while hosting port 8079. Using function:

response = requests.post("{target}/api/function/{variable}/{variable}"

.format(target=backend\_str, v="variable", v= variable))

We have many functions that writes using similar formula, saving licenses, saving car sale details, and other applications, but the main methods for writing are useradd and write\_details which is called within previously mentioned functions.

Write\_details: first creates a folder within our data folder based on the username,

cakepath="data/"+ username + "/"

if not os.path.exists(cakepath):

os.makedirs(cakepath)

of cause checks if a folder of the same name already created, then a file is created using functions

rel\_path = "data/" + name + "/" + name + type

Useradd: is exactly the same as write\_details except Useradd will save a txt file so there will be no type variable, instead a “.txt”, because this file will save more information about the user for further usages, the extra information is written into the file using:

rel\_path = "data/" + username + "/" + username + ".txt"

file = open(rel\_path, mode='w')

file.write('{}${}${}${}'.format(password, email, memberType, TFnumber))

separated using $, and all the extra info are requested variables while calling this function.

***Reading files***: we generally read from dictionary, or simply check if a file exists using .is\_file(), this works by first make a name = path(filepath), then if name.is\_file(), to check if a file is there, here is a good example of how we use this code.

path = "data/" + "honeyport" + "/" + port

my\_file = Path(path)

if my\_file.is\_file():

return True

else:

return False

when we read a user file txt files which saves extra user infos, we generally use userget function from the backend:

@post('/api/userget/<username:path>')

def userget(username):

and the function does is obviously:

if username in user:

rel\_path = "data/" + username + "/" + username + ".txt"

info = open(rel\_path).read()

return info

else:

return "User does not exist!"

checks if user exists, if exists, return the info read from the file, we can use the returned info by first split them using info.split(), could be split(‘$’) since we are saving as $ to space elements and we read what we want example membertype is at position 2, 0-2, then info[2].

**Other important Methods,** our site is using general hash and encrypt method to encrypt important information, adding salt and change formation an such, but we are also using authentication code, which acts as sort of a two factor verification, what it does is first time login, this code will be default to the start and end letter of the username , but every and next time a user login, a new authentication code will be randomly generated for the user, it is basically a 2 letter code containing either a random letter or number. This code will be used next time when the user login, the write and read function for this is the same as others, and it will be displayed as info on the login page after login, we are expecting the user to take notes of this code, and use it next time they login.