Hochschule Offenburg
offenburg.university

Password Management System (PMS)

Assignment

2. Semester

UNITS

Uinclis Levi Martins Seebacher

(187027)

27. June 2021

Processing time	6 months
Surveyor	Prof. Dr. Andreas Schaad

Statutory declaration

I hereby certify under penalty of that I have written this page on the subject of

Password Management System

Has been prepared by me independently and without unauthorized outsized help that I have identified all passages that have been taken verbatim or approximately verbatim or in spirit from publications, unpublished documents, and conversations as such at the appropriate places with the work by means of quotation, whereby the extent of the original quotations taken has been identified in each case in the quotations. The work has not yet been submitted to any examination authority in the same or a similar version and has not yet been published. I am aware that a false statement will have legal consequences.

Freiburg, 20.06.2021

Uinclis

Uinclis Levi Martins Seebacher

Table of Contents

1 Introduction

This is the version 1.0 of the project Password Management System (PMS).

The project PMS was created with the goal for the usage of multiple users at the same time and one manager (admin). The language used for this project was Python 3 for the simple reason to make the usage of third-party providers like Have I Been Pwned (HIBP) and connection via Flask safer and more effective for users and admin.

To make sure that the user data will not be used for inadequate usage the PMS project will **not** store the users' passwords but the hashes of the passwords. That means the admin will not be able to see the users' passwords.

To keep the user even safer a password policy has been created. User's passwords need to fallow following rules:

- ➔ Password needs to contain a specific length.
- ➔ Password can not contain any space.
- ➔ Password needs to contain one or more digits.
- ➔ Password needs to contain lowercases.
- ➔ Password needs to contain uppercases.
- ➔ Password needs to contain punctuations.
- ➔ In case that the user already changed her password, the new password and the old need to be different.

The password criteria are defined by the admin and it can be changed any time only by himself. The admin otherwise is only able to delete accounts of the users and reset it.

For the creation of an account two things are necessary.

- ➔ **A validly username**: That means there are not another user with the same username.
- ➔ **A validly password**: That can be inputted manually by the user or the user has the option to generate a password. It doesn't what the user decides to do a part of the hashed password will be send to HIBP to make sure that the password is 100% safe.

Note: The following diagrams were created together in a learn group with Arin Spinner, Astrit Avdimetaj, Nico Riemer, Nicolas Reinhart und Stefan Götz

2 Requirements

2.1 Use Cases

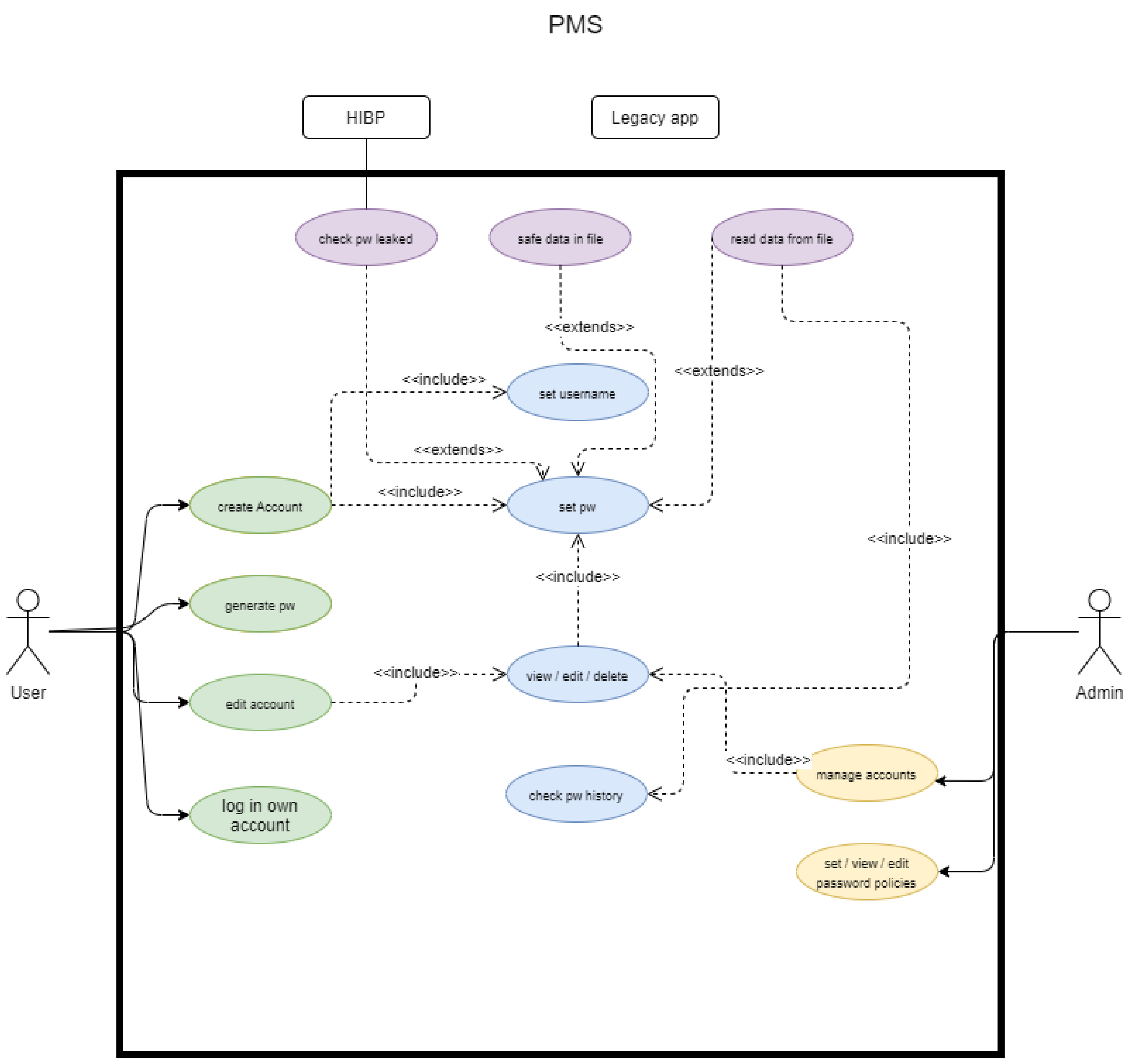


Figure 1 Use Cases

As expected, the use case diagram is showing the entire system in a simple way, as green we can see the decisions that the user can make, as yellow we can see the decisions that the admin can make, as Blue we can see the decision that can be made by the system and as violet we can see the configurations that are saved in the data management. HIBP and Legacy app are both out of the system for the simple fact that they are third-party providers. Here is fundamental to understand the difference between **<<include>>** and **<<extend>>.**

By **<<include>>** means that a case calls another case, and both are necessary

ex: **edit account** includes **view/edit/delete**

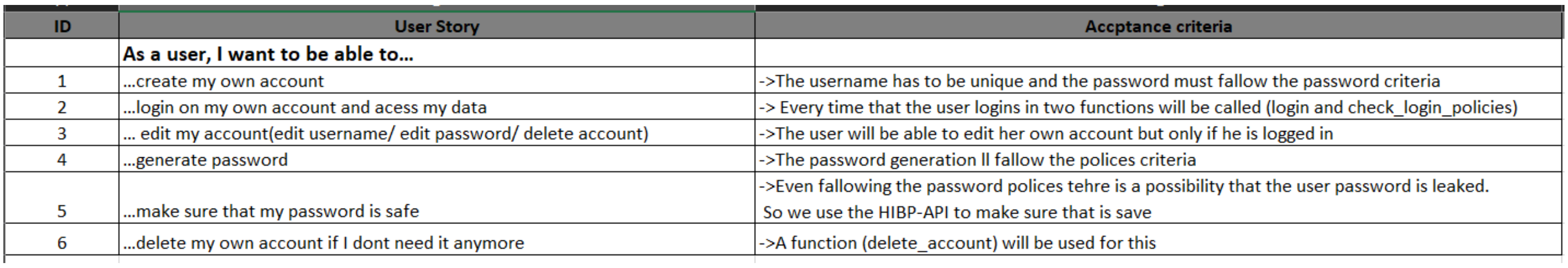
By **<<extend>>** means that a case calls another case, but different of **<<include>>** the case is optional

ex: save data in file extends set pw

By set pw is extended because another case could be called instead of it (check pw/leaked for example).

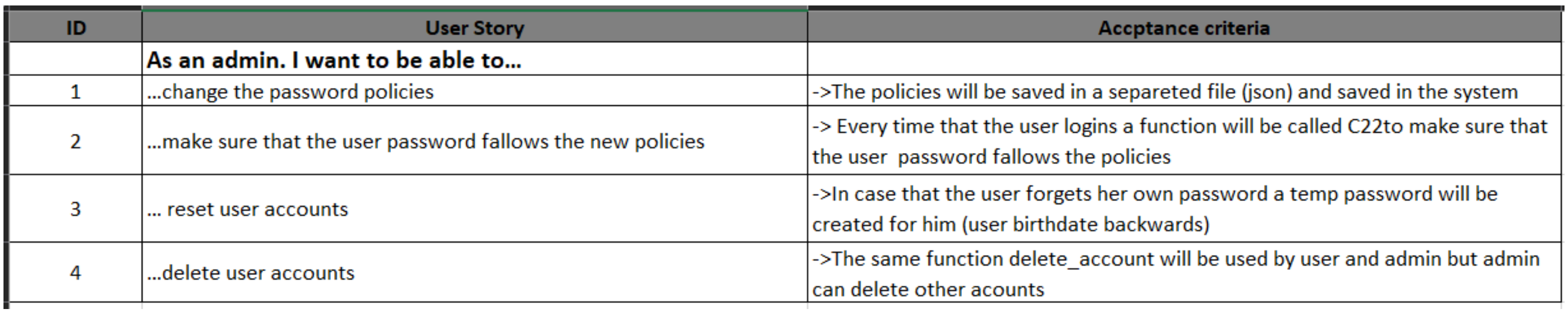
2.1.1 User Stories

User



ID	User Story	Accptance criteria
	As a user, I want to be able to...	
1	...create my own account	->The username has to be unique and the password must fallow the password criteria
2	...login on my own account and acess my data	-> Every time that the user logins in two functions will be called (login and check_login_policies)
3	... edit my account(edit username/ edit password/ delete account)	->The user will be able to edit her own account but only if he is logged in
4	...generate password	->The password generation ll fallow the polices criteria
5	...make sure that my password is safe	->Even fallowing the password polices tehre is a possibility that the user password is leaked. So we use the HIBP-API to make sure that is save
6	...delete my own account if I dont need it anymore	->A function (delete_account) will be used for this

Admin



ID	User Story	Accptance criteria
	As an admin. I want to be able to...	
1	...change the password policies	->The policies will be saved in a separeted file (json) and saved in the system
2	...make sure that the user password fallows the new policies	-> Every time that the user logins a function will be called C22to make sure that the user password fallows the policies
3	... reset user accounts	->In case that the user forgets her own password a temp password will be created for him (user birthdate backwards)
4	...delete user accounts	->The same function delete_account will be used by user and admin but admin can delete other acounts

ID	Type	Description	Related requirements	Unit test ID
S.R. 1	**Non Functional**	**System must manage passwords**		
S.R. 1.2	Functional	A function "set_pw" must be provided		test_set_pw
S.R. 1.3	Functional	A function "check_pw" must be provided		test_check_pw
S.R. 1.4	Functional	A function "gen_pw" must be provided		test_gen_pw_criteria
S.R. 1.5	Functional	A function "hash_pw" must be provided		test_hash_pw
S.R. 1.6	Functional	A function "HIBP_pw" must be provided		test_HIBP_pw
S.R. 1.7	Functional	A function "read_pw" must be provided	S.R. 2.4	test_read_pw
S.R. 1.8	Functional	A function "gen_pw_criteria" must be provided	S.R. 5.1	test_gen_pw_criteria
S.R. 1.9	Functional	A function "old_pw" must be provided		test_old_pw

2.1.2 Functional/Non-Functional

In the requirements table you can see all functions and non-functions that are necessary for the PMS project. But I would like to talk about three functions that in my opinion are fundamental for the usage of the system.

hash_pw

➔ This function has the simple job to turn the password into a hash. That means it receives the valuer user_password as argument and returns a hash.

Ex: This is a password -> 5f8a8eec578c1212980074259dd14518fc9fb249b4473deb2049943a

HIBP_pw

➔ This function sends the hash of own password to the HIBP API and returns a number this number means how many data breaches have this password been found. To make it even safer the function will only send the first 5 digits of the hashed password. Even if the service has been proved to be safe and confidential with the together work with FBI. The first 5 digits is enough for the checking in their data base.

old_pw

➔ Every time when the user changes her password for any reason or the password criteria change the old hashes of the user will be saved and this function will force the user to create a new password different than the old passwords that he already has.

3 High Level Architecture

3.1 TAM

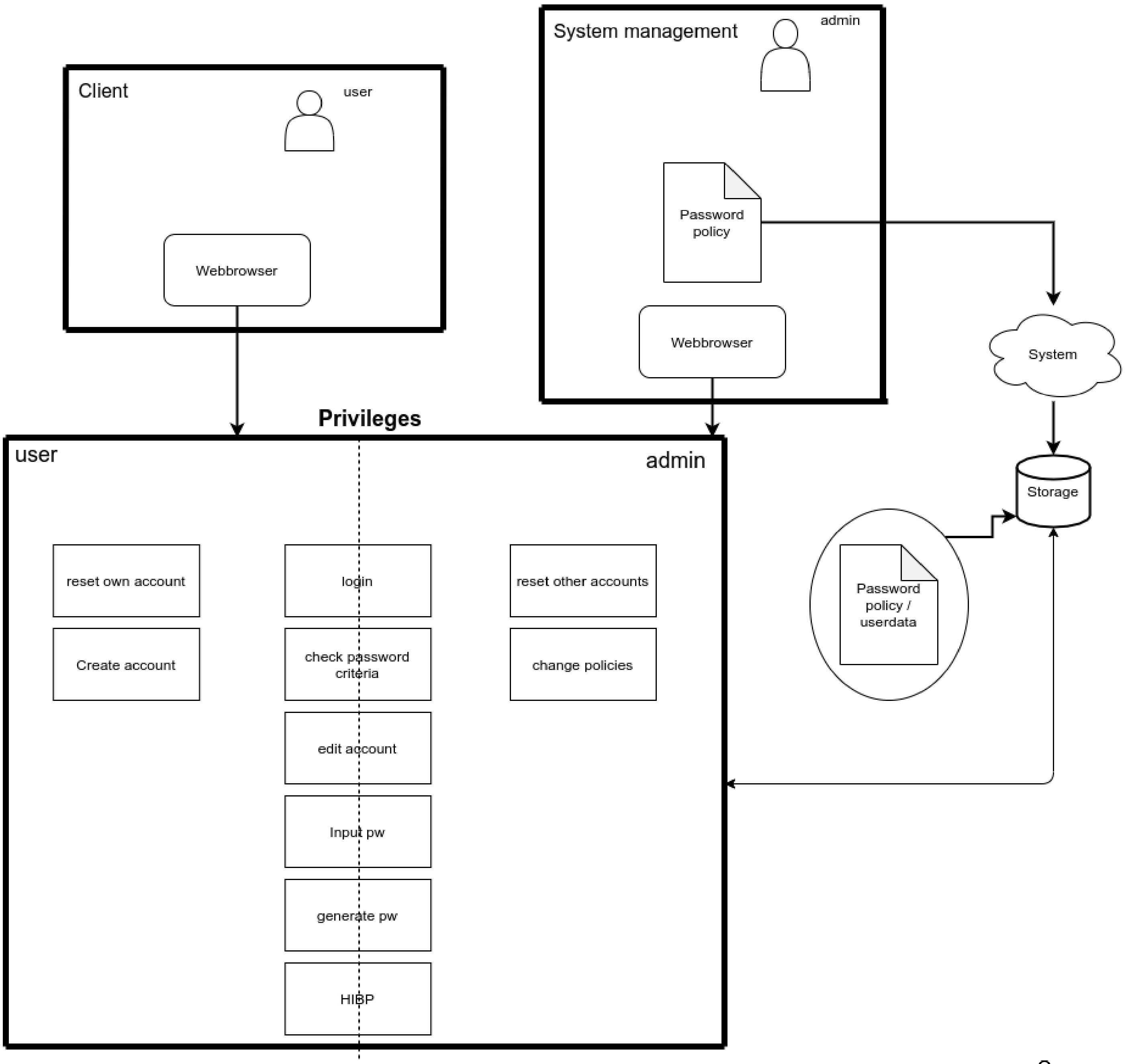


Figure 2 TAM

In the Technical Architecture Modeling (TAM) diagram you can see the overview of the system with the interaction of the user and admin. As you can see it will all happen over web browser. In the privilege box you can see the possibilities that both have. On the left side user privileges, on the right the admin privileges and in the middle the privilege that both have.

4 Design

4.1 Automatically creation of passwords

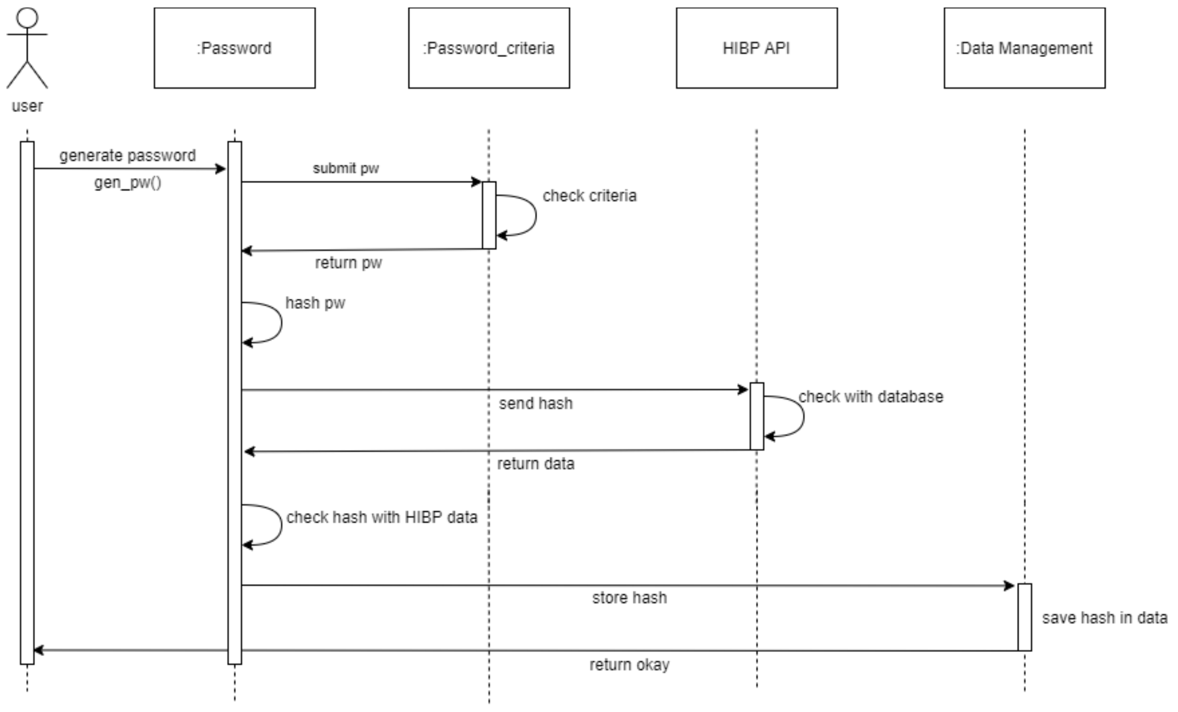


Figure 3 Automatically creation of passwords

In this creation of password diagram, you can see on the top four objects. Three of them have a colon “:” and one doesn’t. That simple means that we have three classes and one third-party provider. The password class is simple there to create password, it can be manually with user input or automatically like in this case. Note that after the password criteria the password will be hashed to keep the date of the user safe. Password criteria is operated under the admin wish that means that the admin can define the criteria for the passwords. HIBP is just there to inform if the hash has been already leaked or not. That means we receive a number as return and in the best case it must be a 0. In the end the hash of own password ll be saved in own data management.

4.2 Manually creation of passwords

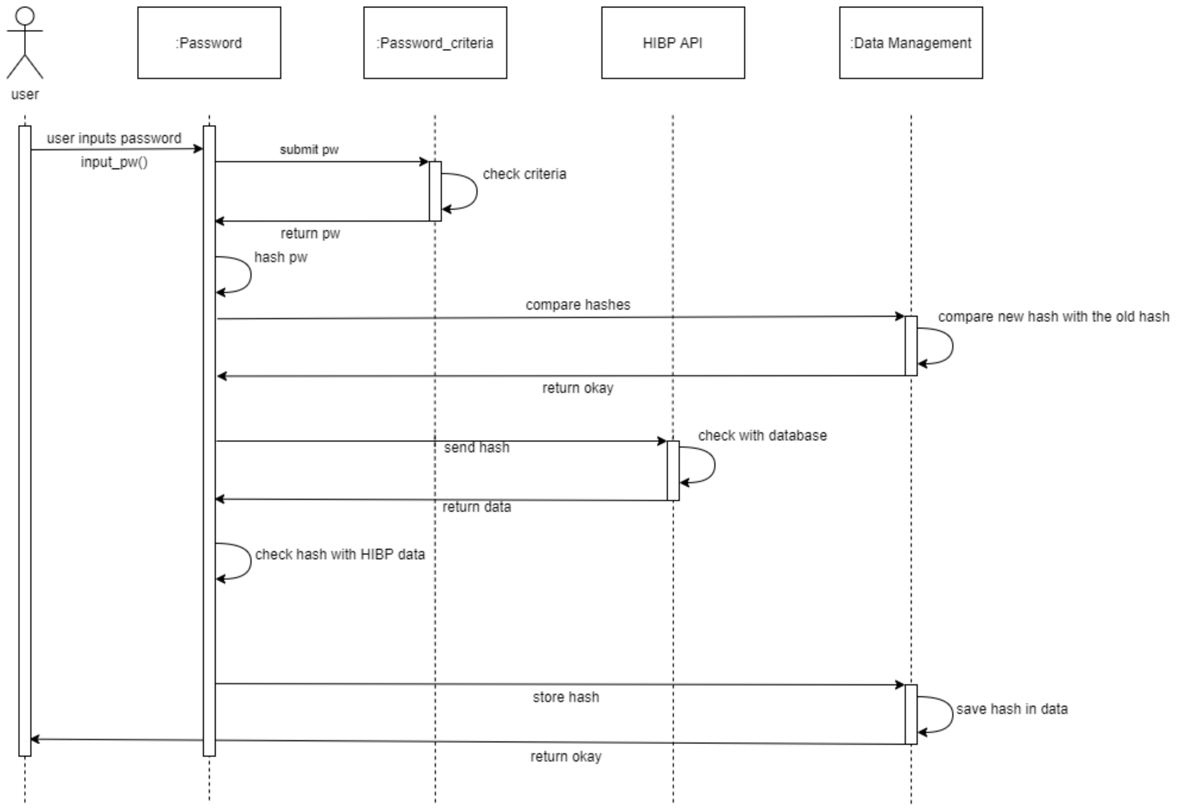


Figure 4 Manually creation of passwords

By manually creation of a password the process is similar to the Automatically creation of password. The only difference is that by Data Management will be accessed twice. First to make sure that the actual password hasn’t been used by the user before and in the second time to save the hashed password in the Data Management.

4.3 Login and creation of accounts

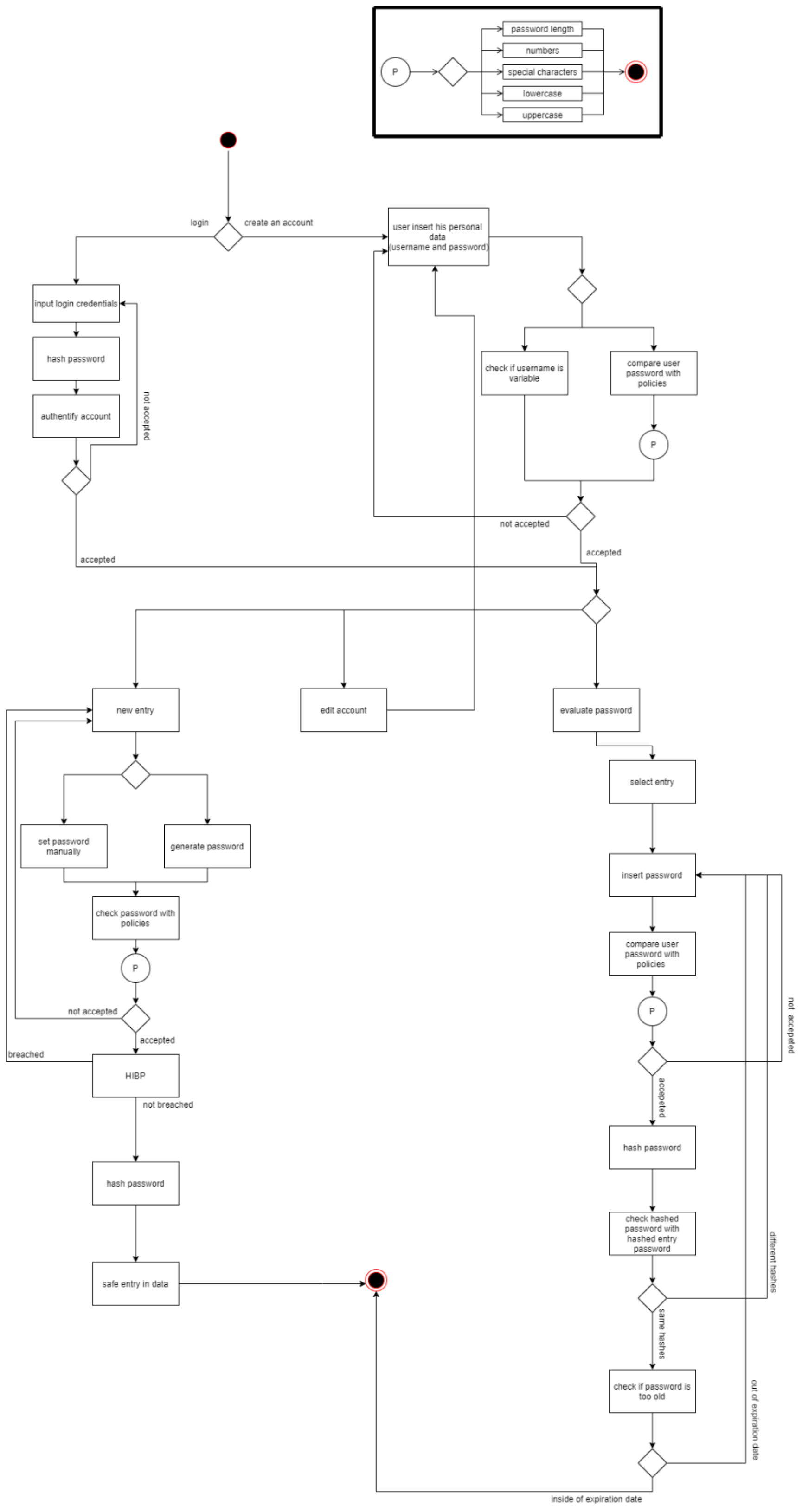


Figure 5 login and creation of accounts

The activity diagram is showing the whole possibilities that the user has for making the interaction with the system possible. Starting with a simple bifurcation where the user can create an account or login if he already has one. In the right side you can see a circle with a P (policies). It is a reference to the top rectangle, it just decided to create it to make the reading of the diagram simpler. In the bottom middle you can see the end of the diagram, marked with a circled red line and a blackhole in the middle.

4.4 Policies update

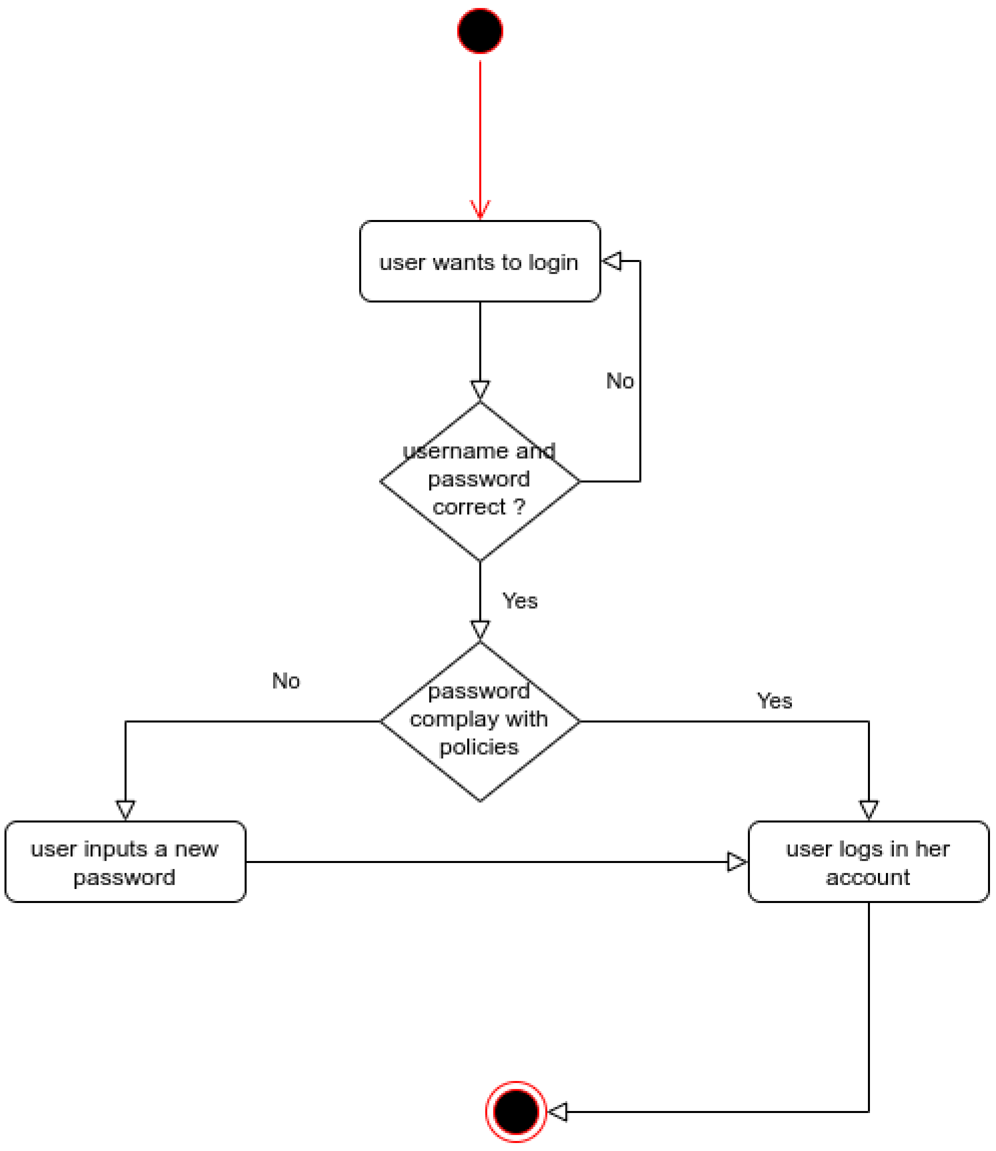

user wants to login
No
username and password correct ?
Yes
No
password complay with policies
Yes
user inputs a new password
user logs in her account

Figure 6 Policies update

As you can see in the diagram. Every time when the admin decides to change the password. The user will not need to logoff, but in the next login their passwords will be checked with the new criteria. That means that there are users which need to change their password and other don't.

ex: admin decides to change the minimal length of the passwords from 8 to 10. Users with a password with a length 10 or longer **don't** need to change their passwords, but users with a password length of 9 or shorter do.

4.5 Class Diagram

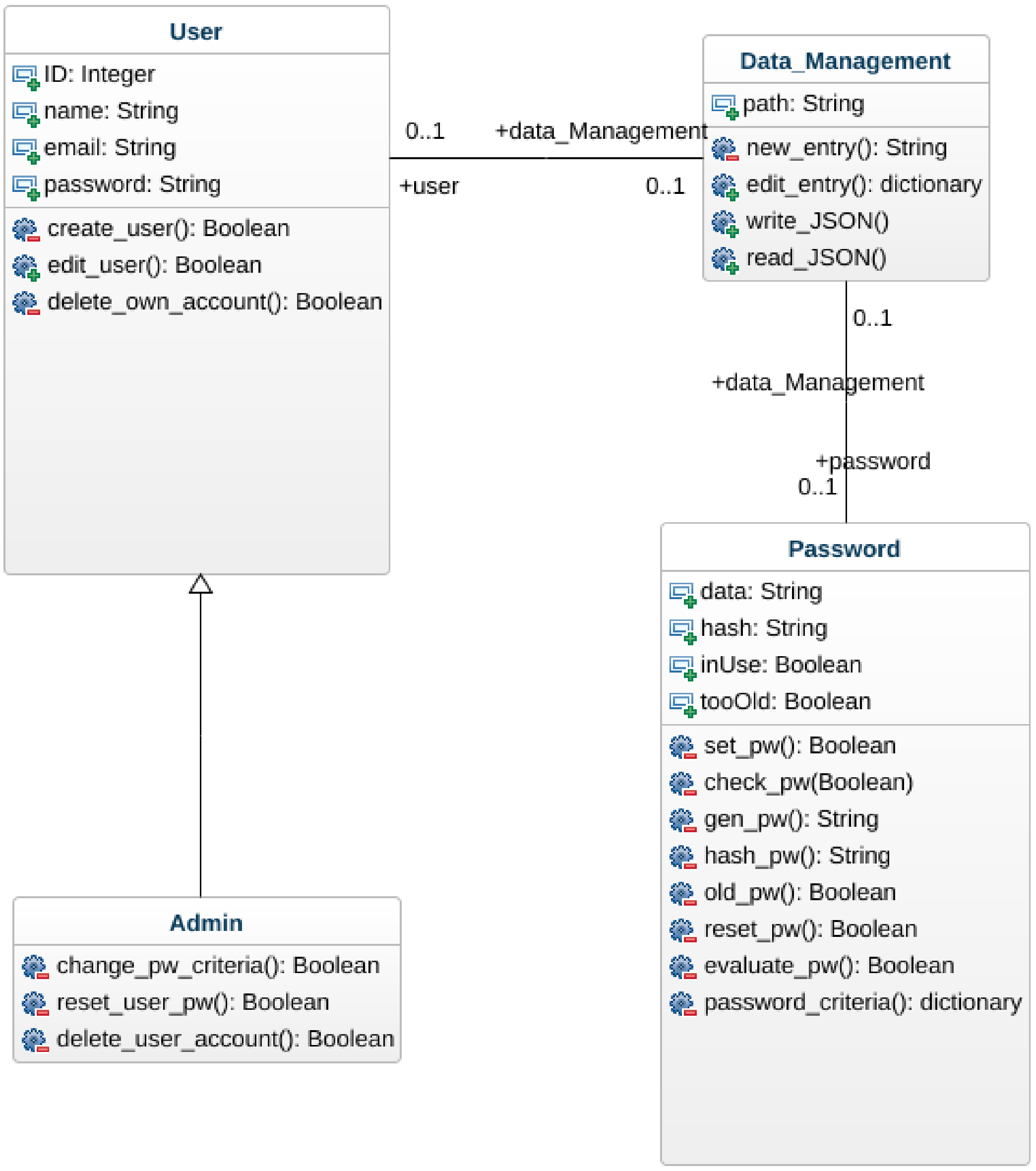


Figure 7 Class Diagram

As you can see in the class diagram there are four classes even if the language chose was Python 3 and the relation of public or private doesn’t matter here, it’s necessary to keep it for the future versions and even if necessary for future projects in another languages like Java.

An important relationship here is the user to admin. Admin class inheritances from the class user and as you can see the admin is nothing more than an user with three more privileges.

change_pw_criteria
→ Admin can change the criteria of the password.

reset_user_pw
→Admin can reset the password of a user. The new temporary password will be the birthdate of the user backwards **12.07.1995** turns into **5991.70.21**.

delete_user_account
→Admin can delete other users accounts if he thinks it is necessary.

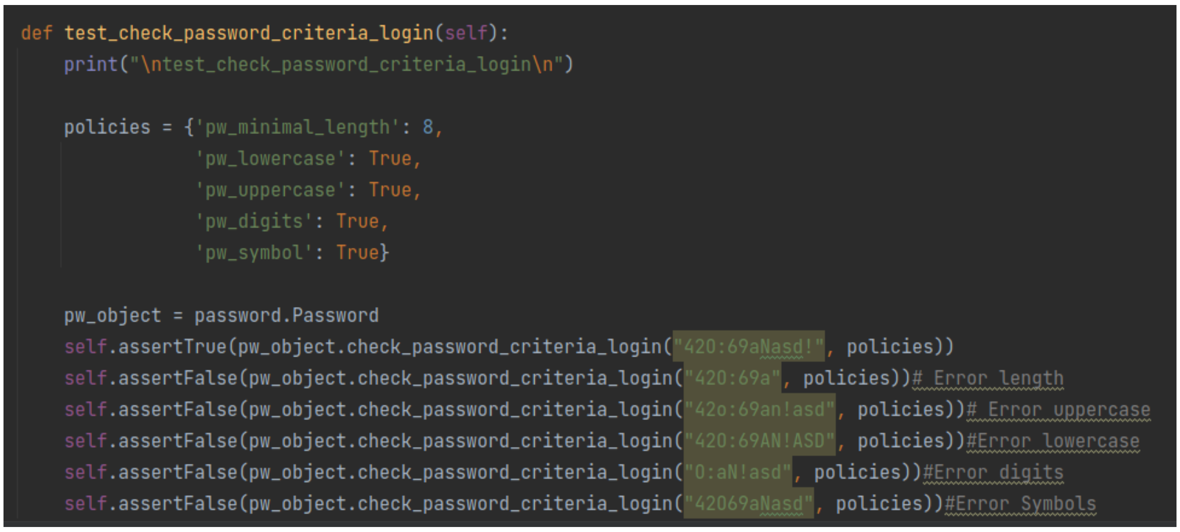
The **Data_Management** class is only there to save in own data of the whole users and admin information.

The **Password** class has the simple responsibility to work with the user passwords: hashing, generating, resetting, checking the input with the hash, checking the old password and more importantly check the password criteria. **password_criteria** is differently than the other functions because it returns a dictionary as return type (password policies).

5 Validation

The validations were made with the help of unittests. As you can see the entire system with class and every class has his own task to complete. Some of the tests are so complex that I decided to print some outputs.

5.1 Test_check_password_criteria_login

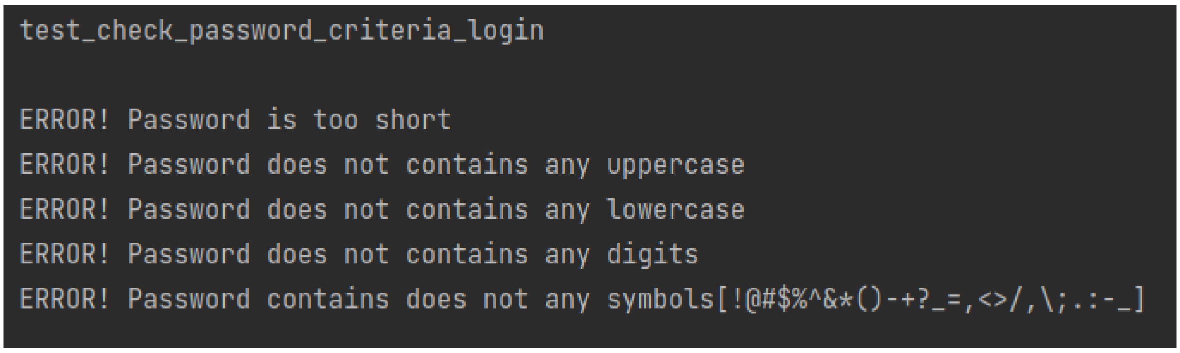


```
def test_check_password_criteria_login(self):
    print("\ntest_check_password_criteria_login\n")

    policies = {'pw_minimal_length': 8,
                'pw_lowercase': True,
                'pw_uppercase': True,
                'pw_digits': True,
                'pw_symbol': True}

    pw_object = password.Password
    self.assertTrue(pw_object.check_password_criteria_login("420:69aNasd!", policies))
    self.assertFalse(pw_object.check_password_criteria_login("420:69a", policies))# Error length
    self.assertFalse(pw_object.check_password_criteria_login("42o:69an!asd", policies))# Error uppercase
    self.assertFalse(pw_object.check_password_criteria_login("420:69AN!ASD", policies))#Error lowercase
    self.assertFalse(pw_object.check_password_criteria_login("0:aN!asd", policies))#Error digits
    self.assertFalse(pw_object.check_password_criteria_login("42069aNasd", policies))#Error Symbols
```

Screenshot 1 test_check_password_criteria_login 1



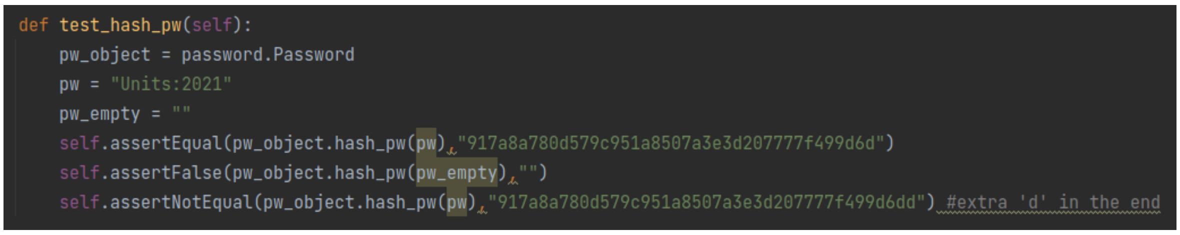
```
test_check_password_criteria_login

ERROR! Password is too short
ERROR! Password does not contains any uppercase
ERROR! Password does not contains any lowercase
ERROR! Password does not contains any digits
ERROR! Password contains does not any symbols[!@#$%^&*()-+?_=,<>/,\;.:-_]
```

Screenshot 2 test_check_password_criteria_login 2

The test_check_password_criteria_login function has the simple task to check the password criteria of the users, every time he logins in. By a policies upgrade must the user change her password before he logs in (in case if the password does not coincide with the password criteria). It receives two inputs *user-password* and *policies* and it can return **TRUE** or **FALSE** (in case of **FALSE** it returns an error message too).

5.2 Test_hash_pw

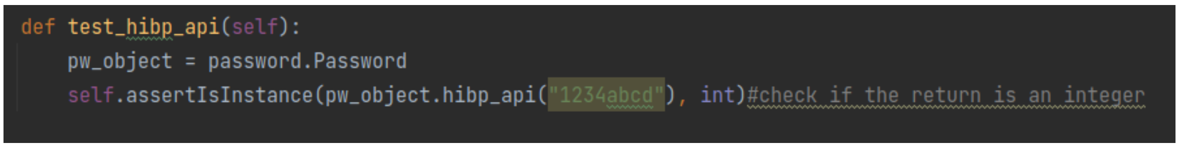


```
def test_hash_pw(self):
    pw_object = password.Password
    pw = "Units:2021"
    pw_empty = ""
    self.assertEqual(pw_object.hash_pw(pw),"917a8a780d579c951a8507a3e3d207777f499d6d")
    self.assertFalse(pw_object.hash_pw(pw_empty),"")
    self.assertNotEqual(pw_object.hash_pw(pw),"917a8a780d579c951a8507a3e3d207777f499d6dd") #extra 'd' in the end
```

Screenshot 3 test_hash_pw

This function otherwise has a simple task to hash a password. It receives a password and returns the same password, but hashed in SHA1. I decided to make three tests to check the output. One of them returns **TRUE,** because that the hash I sent is the same that I was waiting for and in the others, it returns **FALSE** for the simple reason that one test is empty (the variable is empty) and the other the hash is not the same (extra ‘d’ in the end).

5.3 Test_hibp

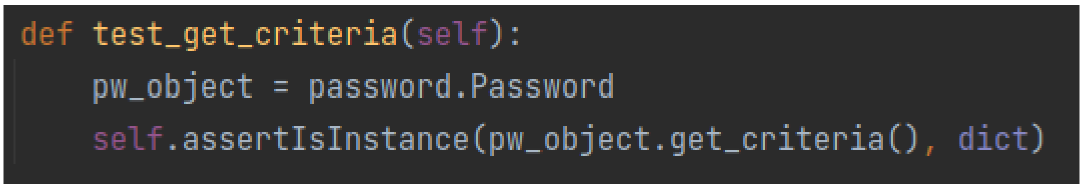


```
def test_hibp_api(self):
    pw_object = password.Password
    self.assertIsInstance(pw_object.hibp_api("1234abcd"), int)#check if the return is an integer
```

Screenshot 4 test_hibp

The function hibp has the simple work to tell me if the datatype that I receive from the HIBP-API is an integer. If the password is safe or not it has the function safe_pw, that is **not** included in the unittests.

5.4 Test_get_criteria

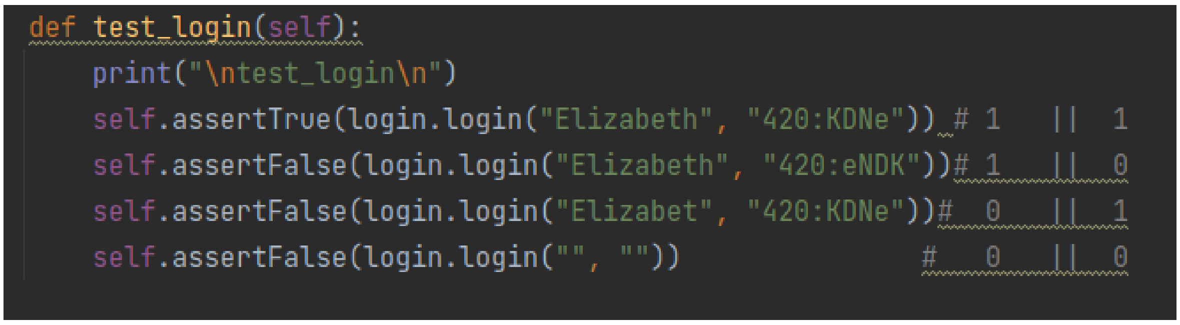


```
def test_get_criteria(self):
    pw_object = password.Password
    self.assertIsInstance(pw_object.get_criteria(), dict)
```

Screenshot 5 test_get_criteria

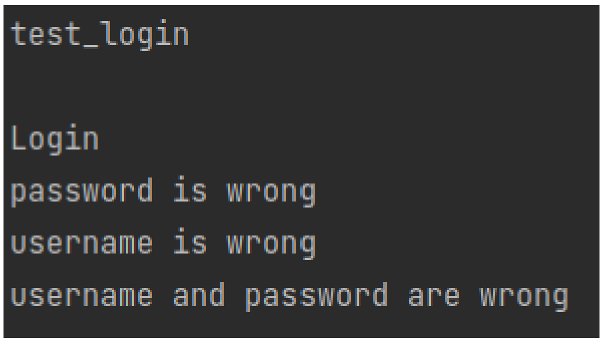
The functions test_get_criteria and test_hibp have the task: check the return valuers. In that case it is a waiting for a dictionary and in the function get_criteria returns **FALSE** in case if the function doesn’t find any Jason file with the name “policies.json”

5.5 Test_login



```
def test_login(self):
    print("\ntest_login\n")
    self.assertTrue(login.login("Elizabeth", "420:KDNe")) # 1   ||  1
    self.assertFalse(login.login("Elizabeth", "420:eNDK"))# 1   ||  0
    self.assertFalse(login.login("Elizabet", "420:KDNe"))#  0   ||  1
    self.assertFalse(login.login("", ""))               #  0   ||  0
```

Screenshot 6 test_login 1



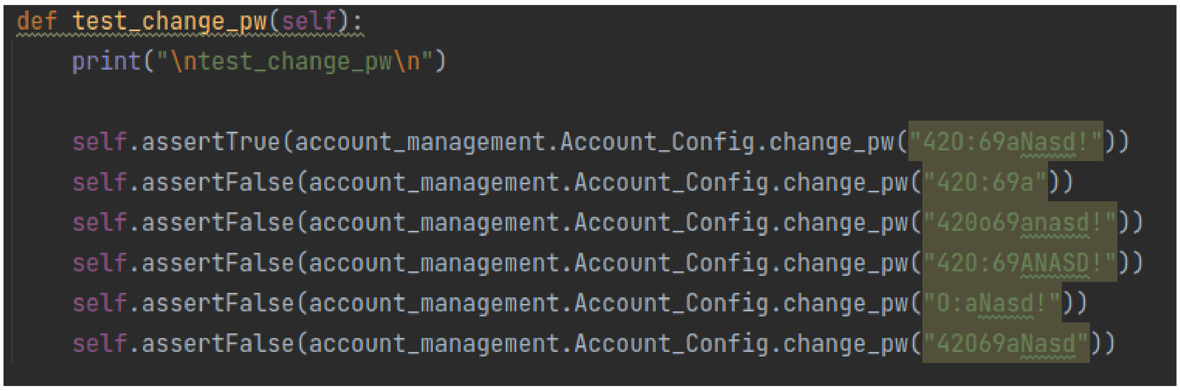
```
test_login

Login
password is wrong
username is wrong
username and password are wrong
```

Screenshot 7test_login 2

The function login receives two valuer’s *username* and *password* and it only returns TRUE when both are correct. That means it looks for the username and password in the data and compares it. If they are right then the function check_password_criteria_login and if the user’s password and the policies match then the user can login in her account. In the example I simple return a error message every time when the user name or password are wrong, but informations like that needs to be changed and cannot be showed.

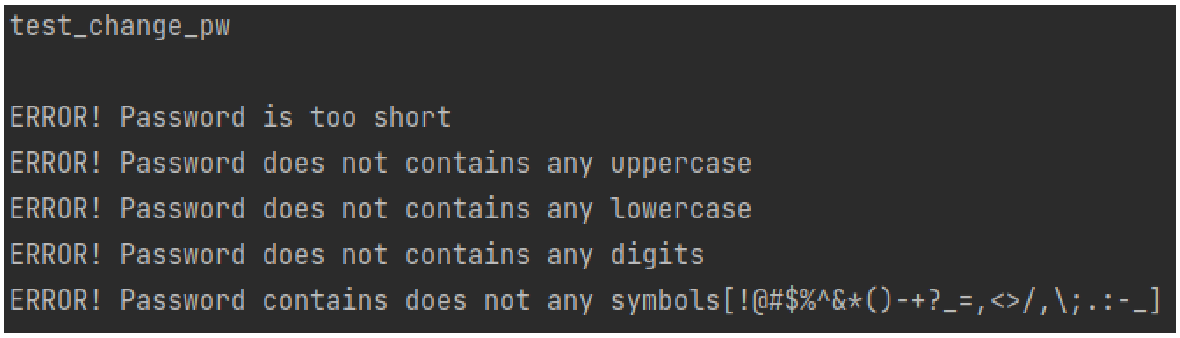
5.6 Test_change_pw



```
def test_change_pw(self):
    print("\ntest_change_pw\n")

    self.assertTrue(account_management.Account_Config.change_pw("420:69aNasd!"))
    self.assertFalse(account_management.Account_Config.change_pw("420:69a"))
    self.assertFalse(account_management.Account_Config.change_pw("420o69anasd!"))
    self.assertFalse(account_management.Account_Config.change_pw("420:69ANASD!"))
    self.assertFalse(account_management.Account_Config.change_pw("O:aNasd!"))
    self.assertFalse(account_management.Account_Config.change_pw("42069aNasd"))
```

Screenshot 8 test_change_pw 1



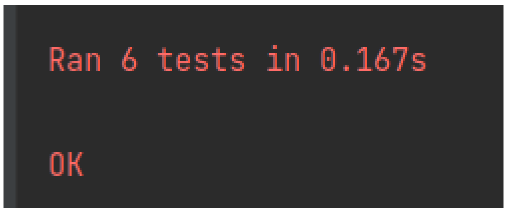
```
test_change_pw

ERROR! Password is too short
ERROR! Password does not contains any uppercase
ERROR! Password does not contains any lowercase
ERROR! Password does not contains any digits
ERROR! Password contains does not any symbols[!@#$%^&*()-+?_=,<>/,\;.:-_]
```

Screenshot 9 test_change_pw 2

The function comes exactly after that the function check_password_criteria_login returns **FALSE** or if the user decides to change her password by her own. Notice here that this time the policies was not used in the test function because it has been called from the class Password in the class Account

5.7 Results



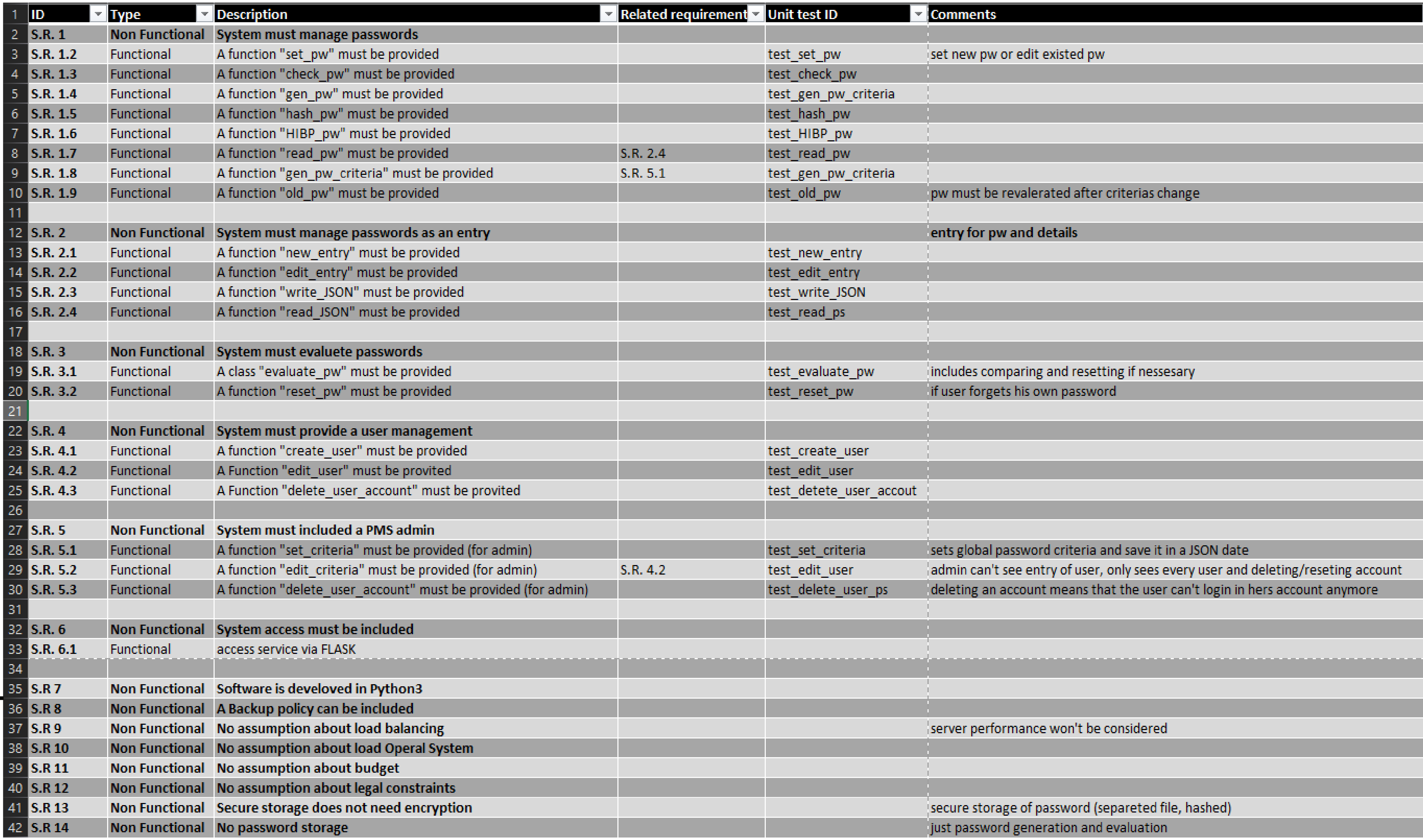
```
Ran 6 tests in 0.167s

OK
```

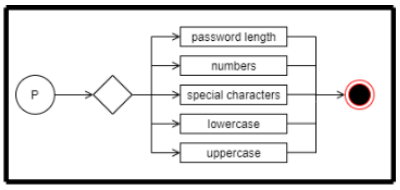
Screenshot 10 Results

As you can see all test were made properly and returned the time and the quantity of tests made to prove that all tests are correct.

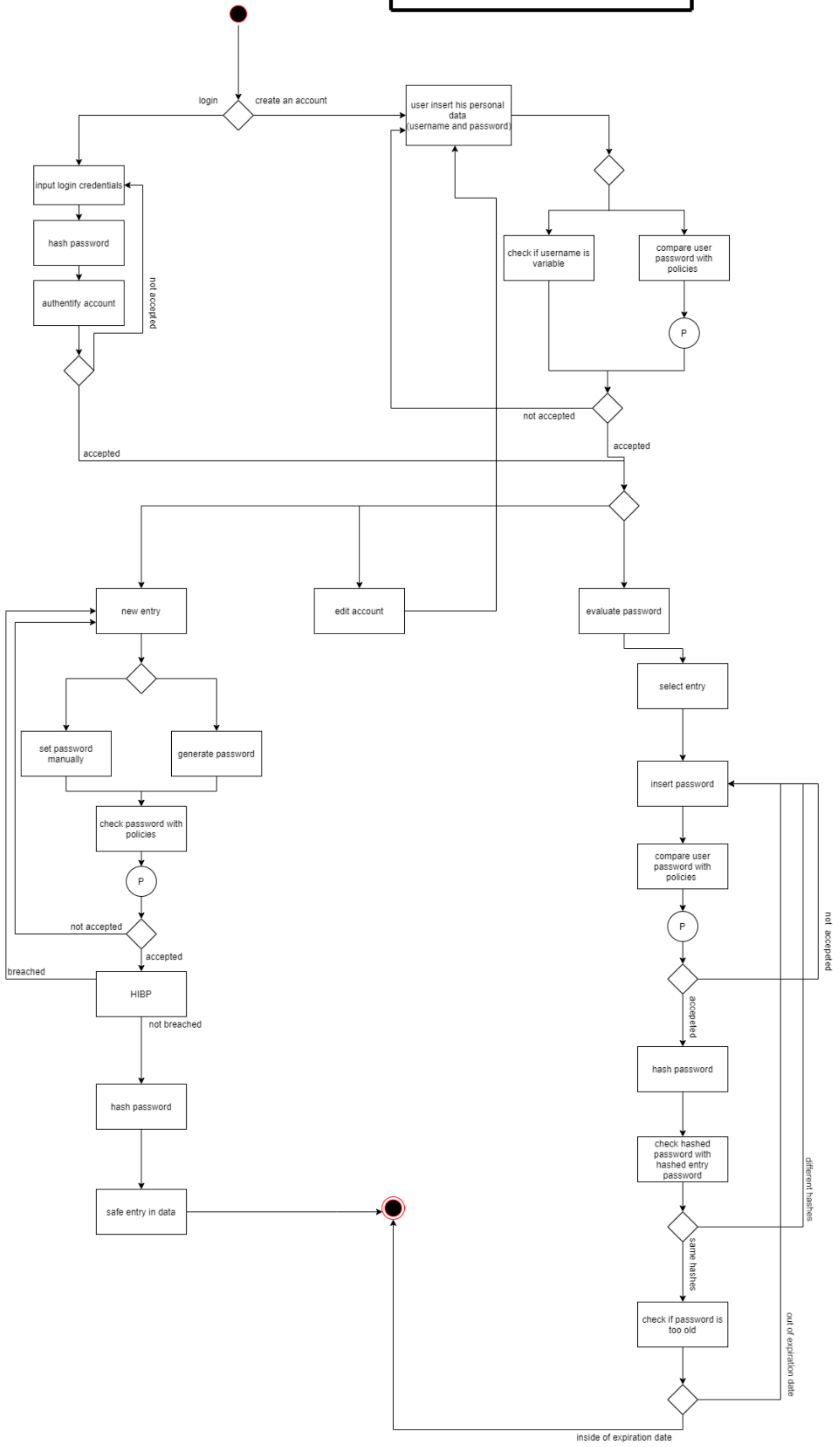
6 Template



	ID	Type	Description	Related requirement	Unit test ID	Comments
2	**S.R. 1**	**Non Functional**	**System must manage passwords**			
3	**S.R. 1.2**	Functional	A function "set_pw" must be provided		test_set_pw	set new pw or edit existed pw
4	**S.R. 1.3**	Functional	A function "check_pw" must be provided		test_check_pw	
5	**S.R. 1.4**	Functional	A function "gen_pw" must be provided		test_gen_pw_criteria	
6	**S.R. 1.5**	Functional	A function "hash_pw" must be provided		test_hash_pw	
7	**S.R. 1.6**	Functional	A function "HIBP_pw" must be provided		test_HIBP_pw	
8	**S.R. 1.7**	Functional	A function "read_pw" must be provided	S.R. 2.4	test_read_pw	
9	**S.R. 1.8**	Functional	A function "gen_pw_criteria" must be provided	S.R. 5.1	test_gen_pw_criteria	
10	**S.R. 1.9**	Functional	A function "old_pw" must be provided		test_old_pw	pw must be revalerated after criterias change
11						
12	**S.R. 2**	**Non Functional**	**System must manage passwords as an entry**			**entry for pw and details**
13	**S.R. 2.1**	Functional	A function "new_entry" must be provided		test_new_entry	
14	**S.R. 2.2**	Functional	A function "edit_entry" must be provided		test_edit_entry	
15	**S.R. 2.3**	Functional	A function "write_JSON" must be provided		test_write_JSON	
16	**S.R. 2.4**	Functional	A function "read_JSON" must be provided		test_read_ps	
17						
18	**S.R. 3**	**Non Functional**	**System must evaluete passwords**			
19	**S.R. 3.1**	Functional	A class "evaluate_pw" must be provided		test_evaluate_pw	includes comparing and resetting if nessesary
20	**S.R. 3.2**	Functional	A function "reset_pw" must be provided		test_reset_pw	if user forgets his own password
21						
22	**S.R. 4**	**Non Functional**	**System must provide a user management**			
23	**S.R. 4.1**	Functional	A function "create_user" must be provided		test_create_user	
24	**S.R. 4.2**	Functional	A Function "edit_user" must be provited		test_edit_user	
25	**S.R. 4.3**	Functional	A Function "delete_user_account" must be provited		test_detete_user_accout	
26						
27	**S.R. 5**	**Non Functional**	**System must included a PMS admin**			
28	**S.R. 5.1**	Functional	A function "set_criteria" must be provided (for admin)		test_set_criteria	sets global password criteria and save it in a JSON date
29	**S.R. 5.2**	Functional	A function "edit_criteria" must be provided (for admin)	S.R. 4.2	test_edit_user	admin can't see entry of user, only sees every user and deleting/reseting account
30	**S.R. 5.3**	Functional	A function "delete_user_account" must be provided (for admin)		test_delete_user_ps	deleting an account means that the user can't login in hers account anymore
31						
32	**S.R. 6**	**Non Functional**	**System access must be included**			
33	**S.R. 6.1**	Functional	access service via FLASK			
34						
35	**S.R 7**	**Non Functional**	**Software is develoved in Python3**			
36	**S.R 8**	**Non Functional**	**A Backup policy can be included**			
37	**S.R 9**	**Non Functional**	**No assumption about load balancing**			server performance won't be considered
38	**S.R 10**	**Non Functional**	**No assumption about load Operal System**			
39	**S.R 11**	**Non Functional**	**No assumption about budget**			
40	**S.R 12**	**Non Functional**	**No assumption about legal constraints**			
41	**S.R 13**	**Non Functional**	**Secure storage does not need encryption**			secure storage of password (separeted file, hashed)
42	**S.R 14**	**Non Functional**	**No password storage**			just password generation and evaluation



P
password length
numbers
special characters
lowercase
uppercase



login
create an account
user insert his personal data (username and password)
input login credentials
hash password
authentify account
not accepted
accepted
check if username is variable
compare user password with policies
P
not accepted
accepted
new entry
edit account
evaluate password
set password manually
generate password
check password with policies
P
not accepted
accepted
breached
HIBP
not breached
hash password
safe entry in data
select entry
insert password
compare user password with policies
P
not accepted
accepted
hash password
check hashed password with hashed entry password
different hashes
same hashes
check if password is too old
out of expiration date
inside of expiration date