1) To store passwords in the Database

```python
# My users table
class users(models.Model):

    username = models.CharField(max_length=255 , primary_key=True)
    password = models.CharField(max_length=255)
    description = models.TextField()

    class Meta:
        db_table = "users"

    def save(self, *args, **kwargs):
        #Here we hash the password before saving it to the database
        self.password = bcrypt.hashpw(self.password.encode(),bcrypt.gensalt())
        super().save(*args, **kwargs)
```

Using Django, the table users (at models.py) of app users , before saving the record to the base, we hash the password field with salting, using the bcrypt library.

2) To record a successful login in the logging table:

```python
# My logging table
class logging(models.Model):
    username = models.CharField(models.ForeignKey("users", on_delete = models.CASCADE) , max_length=255)
    successfulLogin = models.DateTimeField()

    class Meta:
        db_table = "logging"
```

The above code gives us a logging table, with columns: id for primary key, username and timestamp for recording the logs. When the login is successful, the following code creates a logging record and stores it in the logging
table.

```python
logging(username=username,successfulLogin=datetime.datetime.now()).save()
```

3) To “lock” the user after (3) login attempts:

We create a login_attempts.txt file, which contains in each line a string of the form:

username: loginAttempts:lastTimePasswordChanged where:

username: The username of the user (which is stored in the base).

loginAttempts: The number of times a failed login attempt has been made with this username.

lastTimePasswordChanged: For when the user's password was last changed.

We set variable MAX_ATTEMPTS for the number that login attempts should not exceed (line 76).

After checking that the username exists in the users table, we extract the above 3 strings from the function *loggingAttempts(username)* (line 79).

Modifications of the data so that we can read it, from line 84 and below we do

```
75          # Number of allowed login attempts before account is locked
76          MAX_ATTEMPTS = 3
77
78          #userLineNumber => line of user in the login_attempts.txt file(for the writing part)
79          userInfo,userLineNumber = loggingAttempts(username)
80
81          #last password change , we have it as string here
82          userLastPasswordChange = userInfo[2]
83
84          #Convert strings to ints
85          userLogginAttempts = int(userInfo[1])
86          userLineNumber = int(userLineNumber)
```

The function *loggingAttempts(username)* gives us the username:loginAttempts:lastTimePasswordChanged separately in a list of the form: userInfo[]=[username , loginAttempts, lastTimePasswordChanged]

It also returns the number of the line (userLineNumber) where the username we are looking for in login_attempts.txt is located so that we can later increase logginAttempts by 1.

```
29  # File to store login attempts
30  LOGIN_ATTEMPTS_FILE = 'users/login_attempts.txt'
31
32  def loggingAttempts(username):
33      #searchNumber of attempted loggins
34      #after line = line.split(":")
35      #line[0] = username
36      #line[1] = logginAttempts
37      #line[2] = lastTimePasswordChanged
38
39      lineNumber = 0
40      with open(LOGIN_ATTEMPTS_FILE, 'r') as f:
41          lines = f.readlines()
42          for line in lines:
43              line = line.split(":")
44              if line[0] == username:
45                  #return loggin number attempts AND line number of account
46                  return line,lineNumber
47              else:
48                  lineNumber+=1
```

At this point we check if the user has exceeded the maximum allowed number of login attempts.

If so, we write a message that the account has been locked and a redirect is made to the login
page.

```
88          #Check if account is locked (here both passwords are correct)
89          #Save on computations (dont hash etc)
90          if(userLogginAttempts > MAX_ATTEMPTS):
91              messages.error(request,"Login Failed: Account is locked")
92              return redirect("login")
```

The following piece of code is executed when the user has given the correct username but the wrong password, so:

1) We increment userLogginAttempts by 1 (line 129)
2) We read the data of the loggin_attempts.txt file as a list (line 132-133)
3) We change the line containing the user with the username we got from login, with the new value userLogginAttempts. (line 136)
4) We write the data again in loggin_attempts.txt (lines 139-140)

```
128
129             userLogginAttempts += 1
130
131             #Get data (to change logging number)
132             with open(LOGIN_ATTEMPTS_FILE, 'r') as f:
133                 data = f.readlines()
134
135             #Change data to correct
136             data[userLineNumber] = f"{username}:{userLogginAttempts}:{userLastPasswordChange}"
137
138             #Write data to txt file
139             with open(LOGIN_ATTEMPTS_FILE, 'w') as f:
140                 f.writelines(data)
```

Finally, we check if it is the 3rd time the user makes a login attempt (line 143).

If so, we send a message that the account has been locked and we redirect him to the login page, otherwise we tell him that he wrote the wrong username or password and we redirect him to the login page.

```
142             ##---CHECK IF USER LOGGING NUMBER > MAX_ATTEMPTS---##
143             if(userLogginAttempts > MAX_ATTEMPTS):
144                 messages.error(request,"Exceeded login retry limit. Account is now locked")
145                 return redirect("login")
146             ##--------------------END--------------------------##
147
148             ##-----------------------------------------##
149             ##-----INCREMENT USER LOGGING NUMBER----------##
150             ##----------------END----------------------##
151
152             messages.info(request,"Login Failed: Your username or password is incorrect")
153             return redirect("login")
154
```

4) Asking the user to change his password:

In line 161 we set the days after the last password change that must pass to suggest the user to change the password.

In line 165 we convert the string userLastPasswordChange to a date() object.

In line 167 we calculate the date() date after which the code will have expired. (that is, they will have passed since the date of the last password change).

In line 169 we set an empty string , and we will write in it if the date of line 167 has passed from today's (line 170).

We will then write in the message, after a successful login, that the password must be changed.

```
160     #Days since the last change of the password
161     PASSWORD_EXPIRATION_INTERVAL = 90
162
163     #Convert date string to date (of userLastPasswordChange variable)
164     #userLastPasswordChange[:-2] => the -2 to remove the string /n
165     userLastPasswordChange = datetime.datetime.strptime(userLastPasswordChange[:-2], '%Y-%m-%d').date()
166
167     passwordExpirationTime = userLastPasswordChange + datetime.timedelta(days=PASSWORD_EXPIRATION_INTERVAL)
168
169     changePasswordMessage = ""
170     if datetime.datetime.now().date() > passwordExpirationTime:
171         changePasswordMessage = f"Change of password is recommended!"
172
```