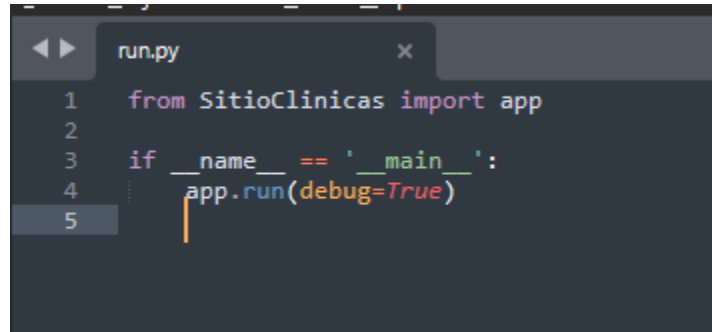


Manual de usuario del código

En este manual se explicará el funcionamiento del código por partes. El código está dividido principalmente en 3 partes: run.py que es el que se encarga de correr el código, _init_.py en donde se inicializan variables y prueba.py donde están las funciones y direcciones de la página web.

Run.py



```
run.py
1  from SitioClinicas import app
2
3  if __name__ == '__main__':
4      app.run(debug=True)
5
```

Como se menciona al principio, al ejecutar este archivo pone en funcionamiento la página web. La página podrá ser vista en la dirección <http://127.0.0.1:5000>. En caso de que se necesite ver la página en la red se puede agregar "host=0.0.0.0" en los parámetros.

Init.py

```
__init__.py  Prueba.py

1  from flask import Flask
2  from flask_login import LoginManager
3  from flask_mail import Mail
4
5  app = Flask(__name__)
6
7  login_manager=LoginManager(app)
8  app.config['SECRET_KEY']='0896709031ff17d407ba27a6ad0dd96e'
9  app.config['MAIL_SERVER']='smtp.gmail.com'
10 app.config['MAIL_PORT']=587
11 app.config['MAIL_USE_TLS']=True
12 app.config['MAIL_USERNAME']='adrimarvi16@gmail.com'
13 app.config['MAIL_PASSWORD']='rbda fcd b ydvb luza'
14
15 mail=Mail(app)
16
17 from SitioClinicas import Prueba
```

En este archivo importamos librerías de Flask que utilizaremos e inicializamos algunos elementos como lo es app. Posteriormente indicamos el valor de la secret key y valores para poder mandar correos electrónicos. En el caso de Mail_Username se debe introducir un correo de Gmail y como Mail_Password una contraseña de aplicación por cuestiones de seguridad de la cuenta de Gmail. <https://support.google.com/accounts/answer/185833?hl=es>

Finalmente inicializamos mail e importamos el archivo Prueba.

Forms.py

```
Forms.py  agregarSuperAdmin.py  Prueba.py

1  from flask_wtf import FlaskForm
2  from flask_wtf.file import FileAllowed, FileField
3  from wtforms import StringField, PasswordField, SubmitField, BooleanField, TextAreaField, SelectField, DateField, IntegerField
4  from wtforms.validators import DataRequired, Length, Email, EqualTo, ValidationError, NumberRange, Regexp, Optional
5
6  class registrarAdmin(FlaskForm):
7      clinica=SelectField('Clínica', choices=[('Odontologia','Odontologia'),('Fisioterapia','Fisioterapia'),('Optometria','Optometria')])
8      id=StringField('id', validators=[DataRequired(), Length(min=2,max=40)])
9      nombre=StringField('Nombre', validators=[DataRequired(), Length(min=2,max=40)])
10     correo=StringField('Correo electrónico', validators=[DataRequired(), Email(message='El correo electrónico no es valido')], Length(min=2,max=40))
11     username=StringField('Usuario', validators=[DataRequired(), Length(min=2,max=20)])
12     password=PasswordField('Contraseña', validators=[DataRequired()])
13     confirm_password=PasswordField('Confirmar contraseña', validators=[DataRequired(), EqualTo('password')])
14     submit=SubmitField('Registrar Admin')
15
16     class registrarProfesor(FlaskForm):
17         clinica=SelectField('Clínica', choices=[('Odontologia','Odontologia'),('Fisioterapia','Fisioterapia'),('Optometria','Optometria')])
18         rfc=StringField('rfc', validators=[DataRequired(), Length(min=2,max=40)])
19         nombre=StringField('Nombre', validators=[DataRequired(), Length(min=2,max=40)])
20         apellidos=StringField('Apellidos', validators=[DataRequired(), Length(min=2,max=40)])
21         correo=StringField('Correo electrónico', validators=[DataRequired(), Email(message='El correo electrónico no es valido')], Length(min=2,max=40))
22         username=StringField('Usuario', validators=[DataRequired(), Length(min=2,max=40)])
23         password=PasswordField('Contraseña', validators=[DataRequired()])
24         confirm_password=PasswordField('Confirmar contraseña', validators=[DataRequired(), EqualTo('password')])
25         submit=SubmitField('Registrar Profesor')
```

En este archivo se encuentran los forms que se ocuparán en prueba.py. Se utilizó FlaskForm para la creación de las clases, flask_wtf.file para el manejo de archivos, wtforms para los tipos de datos y wtforms.validators para realizar las validaciones. Hay 12 clases para diferentes casos de usos.

Para las clases de fisioterapia y odontología se reutilizo información de la clase de optometría ya que aún falta la información de dichas clínicas.

User.py

```
User.py  ModelUser.py

1  from werkzeug.security import check_password_hash
2  from flask_login import UserMixin
3
4  #Se crea una clase User que contiene id, username, password, rol y clinica
5  class User(UserMixin):
6
7      def __init__(self, id, username, password, rol="", clinica="") -> None:
8          self.id = id
9          self.username = username
10         self.password = password
11         self.rol = rol
12         self.clinica = clinica
13
14         #Se crea una función regresa True o False dependiendo de si la contraseña
15         #introducida coincide con la guardada en la base de datos
16         @classmethod
17         def check_password(self, hashed_password, password):
18             return check_password_hash(hashed_password, password)
```

En este archivo se crea una clase User que contiene id, username, password, rol y clínica. También contiene una función que regresa True o False dependiendo de si la contraseña introducida coincide con la guardada en la base de datos utilizando check_password_hash de werkzeug.security.

ModelUser.py

```
ModelUser.py  Prueba.py  x
1  from .entities.User import User
2
3  #Se crea una clase ModelUser
4  class ModelUser():
5
6      #Contiene una función que obtiene la información de dicho usuario en la base de datos para poder iniciar sesión
7      @classmethod
8      def login(self, db, user):
9          try:
10             print("voy a conectarme a la db")
11             cursor = db.cursor()
12             print("me conecte a la db")
13             sql = """SELECT id, username, contraseña, rol, clinica FROM usuario
14                     WHERE username = '{}'.format(user.username)
15             cursor.execute(sql)
16             print("ejecute la SQL")
17             row = cursor.fetchone()
18             if row != None:
19                 user = User(row[0], row[1], User.check_password(row[2], user.password), row[3], row[4])
20                 return user
21             else:
22                 return None
23         except Exception as ex:
24             raise Exception(ex)
```

En la primera parte del código llamamos a User y creamos una clase ModelUser que contiene una función para obtener información del usuario en la base de datos para poder iniciar sesión.

```
25
26  #Contiene una función que obtiene la información a través del id del usuario en la base de datos
27  @classmethod
28  def get_by_id(self, db, id):
29      try:
30          cursor = db.cursor()
31          sql = "SELECT id, username, rol, clinica FROM usuario WHERE id = {}".format(id)
32          cursor.execute(sql)
33          row = cursor.fetchone()
34          if row != None:
35              return User(row[0], row[1], None, row[2], row[3])
36          else:
37              return None
38      except Exception as ex:
39          raise Exception(ex)
```

La segunda función nos permite obtener información del usuario, una vez que ha iniciado sesión, a través de su id.

Prueba.py

```
1 from flask import render_template, url_for, flash, redirect, request, abort, current_app, session
2 from SitioClinicas import app, mail
3 from SitioClinicas.Forms import registrarAdmin, registrarProfesor, registrarAlumno, registrarPaciente, consulta, consultaPaciente
4 from flask_login import LoginManager, login_user, current_user, logout_user, login_required
5 from flask_principal import Principal, Identity, AnonymousIdentity, identity_changed, identity_loaded, RoleNeed, UserNeed, Permission
6 import mysql.connector
7 from werkzeug.security import generate_password_hash
8 from werkzeug.utils import secure_filename
9 from flask_mail import Message
10 from itsdangerous import TimedJSONWebSignatureSerializer as Serializer
11 import os
12 from datetime import date, datetime
13 from apscheduler.schedulers.background import BackgroundScheduler
14 import atexit
15 from models.ModelUser import ModelUser
16 from models.entities.User import User
17
```

Lo primero que se hace en este archivo es importar las librerías y funciones que ocuparemos:

- Flask: Para el funcionamiento de la página web.
- SitioClinicas: Llama las variables creadas en el archivo `_init_.py`.
- SitioClinicas.Forms: Llama las clases de forms que ocuparemos.
- Flask_login: Para manejar el inicio y cerrado de sesión, así como restringir el acceso.
- Flask_principal: Para manejar los permisos otorgados por los roles de admin, profesor y alumno.
- mysql.connector: Para permitir la conexión a la base de datos.
- Werkzeug.security: Para convertir las contraseñas ingresadas en un hash.
- Werkzeug.utils: Para confirmar la seguridad en los nombres de los archivos.
- Flask.mail: Para permitir el envío de correos electrónicos.
- Itsdangerous: Para permitir la creación tokens con caducidad.
- Os: Para el manejo de rutas de archivos
- Datetime: Para el manejo de fechas
- Apscheduler: Para la creación de recordatorios automáticos.
- Models.ModelUser: Para el uso de la clase ModelUser
- Models.entities.User: Para el uso de la clase User.

```
19 # Configura los detalles de la conexión a la base de datos
20 db_config = {
21     'user': 'root',
22     'password': 'contraseña',
23     'host': 'localhost',
24     'database': 'serviciosocial',
25 }
26
27 Principal(app)
28
29 login_manager_app = LoginManager(app)
```

El siguiente paso es ingresar la información de la base de datos para poder conectarse, en caso de que se cambie la información, estos datos deberán ser modificados.

También se inicializa Principal para los permisos y LoginManager para manejar el inicio de sesión.

```
run.py x Prueba.py
31 #Función para manejar la sesión con el usuario
32 @login_manager_app.user_loader
33 def load_user(id):
34     conn = mysql.connector.connect(**db_config)
35     return ModelUser.get_by_id(conn, id)
36
37 #Función niega la entrada a persona sin iniciar sesión
38 @login_manager_app.unauthorized_handler
39 def unauthorized():
40     return redirect(url_for('hello_world'))
41
42 #Función para manejar la identidad del usuario
43 @identity_loaded.connect_via(app)
44 def on_identity_loaded(sender, identity):
45     # Asigna el objeto user a la identidad
46     identity.user = current_user
47
48     # Agrega el UserNeed a la identidad
49     if hasattr(current_user, 'id'):
50         identity.provides.add(UserNeed(current_user.id))
51
52     # Se agrega el rol a la identidad
53     if hasattr(current_user, 'rol'):
54         identity.provides.add(RoleNeed(current_user.rol))
55
56 #Crea los permisos de acceso
57 superAdmin_permission = Permission(RoleNeed('SuperAdmin'))
58 admin_permission = Permission(RoleNeed('SuperAdmin'), RoleNeed('Admin'))
59 profesor_permission = Permission(RoleNeed('SuperAdmin'), RoleNeed('Admin'), RoleNeed('Profesor'))
60
```

La función de la línea 32 obtiene información necesaria para manejar la sesión del usuario en la página. La siguiente función redirecciona a la pagina de iniciar sesión en caso de que alguien intente acceder a paginas que requieran tener una sesión activa.

La siguiente función maneja la identidad del usuario utilizando el id como referencia y el rol para determinar los permisos otorgados. Finalmente se crean los permisos para el SuperAdmin, admin y profesores.

```
62 #Se crea la función que checa que haya información en el dato de "Proxima visita" en las tablas de optometría, fisioterapia y odontología
63 #En caso que falten 14 días, se mandará un correo con el id y nombre del paciente así como la fecha de la visita
64 def recordatorio():
65     with app.app_context():
66
67         #Recordatorio para optometria, hace un join con las tablas de optometría y usuario para obtener la información
68         conn = mysql.connector.connect(**db_config)
69         cur = conn.cursor()
70         cur.execute("""SELECT opt.idPaciente, opt.proximaVisita, pac.nombrePaciente, pac.apellidoPaternoPaciente, pac.apellidoMaternoPaciente
71             from optometria as opt left join paciente as pac on opt.idPaciente=pac.idPaciente WHERE opt.proximaVisita IS NOT NULL""")
72         data = cur.fetchall()
73         cur.close()
74         conn.close()
75         lista=[]
76         newline=""
77
78         #Se lee la información recabada y verifica que falten 14 días, en caso de que asi sea, el paciente se agrega a una lista
79         for visita in data:
80             today = date.today()
81             new=datetime.strptime(visita[1], '%Y-%m-%d')
82             difference=new.date()-today
83             if difference.days==14:
84                 lista.append(visita)
```

La siguiente función es el recordatorio para mandar un correo en caso de que falten 14 días para la cita. Utilizando app.app_context(), nos conectamos a la base de datos para obtener información de la clínica de optometría y del paciente mediante

un left join. Revisamos la información obtenida y verificamos si la próxima visita es en 14 días, en caso de que se cumpla la condición, el paciente es agregado a una lista.

```
185
186     #Si la lista no esta vacia, se le da formato y se manda al correo pertinente
187     #El "recipient" debera ser cambiado por el correo deseado
188     if lista:
189         msg=Message('Recordatorio próximas citas Optometría', recipients=['adrimarvii@hotmail.com'], sender='noreply@hotmail.com')
190         msg.body=f'Los siguientes pacientes tienen una visita el {lista[0][1]}:\n\n{newline.join(f'Id: {value[0]} Nombre: {value[2]} {'
191
192         mail.send(msg)
193         print('Mensaje enviado')
194
```

Si la lista no esta vacía, se le da formato poniendo todos los pacientes en la lista y se envía el correo electrónico.

Este proceso se repite para las clínicas de fisioterapia y odontología.

```
152
153 ▼ with app.app_context():
154     scheduler = BackgroundScheduler()
155     scheduler.add_job(func=recordatorio, trigger="cron", hour=12)
156     scheduler.start()
157
158 # Apagar el scheduler cuando se sale de la app
159 atexit.register(lambda: scheduler.shutdown())
160
```

Finalmente se inicializa el scheduler, se le agrega la función recordatorio, se le indica que lo haga diario a las 12 P.M. y se le indica que comience. El ultimo renglón es para apagar el scheduler al salir de la app.

```
161 #Función que se realiza al acceder a la pagina principal http://127.0.0.1:5000/, es la página de inicio de sesión
162 @app.route("/", methods=['GET', 'POST'])
163 def hello_world():
164     #Si el usuario ya tiene una sesión activa, se redirecciona a la página principal
165     if current_user.is_authenticated:
166         return redirect(url_for('inicio'))
167
168     #Al ser la página de inicio de sesión, el form sera el de login
169     form=LoginForm()
170
171     #Si se hace un request, se crea un user con el username y contraseña ingresado y se utiliza la función ModelUser.Login
172     #Para buscar el username y si existe, comparar la contraseña ingresada con la guardada en la base de datos
173     if form.validate_on_submit():
174         user=User(0, username=form.username.data, password=form.password.data)
175         conn = mysql.connector.connect(**db_config)
176         logged_user = ModelUser.login(conn, user)
177
```

La siguiente función se muestra al acceder a la pagina de inicio de sesión, si el usuario ya esta con una sesión activa, se muestra la página de inicio.

El form que se utiliza es el de login. Si se hace un request se crea un user con el username y contraseña ingresados, y se utiliza ModelUser.Login para comparar las contraseñas y buscar al usuario.

```

177
178     #Si el usuario existe y las contraseñas coinciden, se inicia sesión y se crea la identidad
179     if logged_user!=None:
180         if logged_user.password:
181             flash("Sesión iniciada",'success')
182             login_user(logged_user)
183             identity_changed.send(current_app._get_current_object(), identity=Identity(logged_user.id))
184
185         else:
186             #Si la contraseña es incorrecta, redirecciona a la misma página de inicio de sesión y se le indica al usuario
187             flash("Contraseña inválida... ", 'danger')
188             return render_template('login.html',form=form)
189
190         else:
191             #Si la usuario no se encuentra, redirecciona a la misma página de inicio de sesión y se le indica al usuario
192             flash("No se encuentra el usuario... ", 'danger')
193             return render_template('login.html', form=form)
194             return redirect(url_for('inicio'))
195
196     # Si no se envió ningún form, simplemente muestra la página de inicio de sesión
197     return render_template('login.html', form=form)

```

Si son iguales y el usuario existe, se inicia sesión y se crea la identidad. En caso contrario se le indica al usuario y se mantiene en la misma página.

Al cargar la página, el template que se muestra es el de login.html.

```

198 #En la página de inicio, unicamente se muestra el template de Inicio.html
199 @app.route("/inicio")
200 @login_required
201 def inicio():
202     return render_template('Inicio.html')
203

```

En la página de inicio únicamente se muestra el template de Inicio.html

```

203
204 #En esta función registraremos admins, se requiere tener una sesión activa y tener el rol de SuperAdmin
205 @app.route("/registrarAdmin", methods=['GET','POST'])
206 @login_required
207 @superAdmin_permission.require(http_exception=403)
208 def registrarAdmin():
209     #Se usará el form registrarAdmin
210     form=registrarAdmin()
211
212     #Si se hace un request, la contraseña se convierte en un hash por seguridad y se checa en la tabla usuario si el
213     #usuario o contraseña que se esta registrando, ya han estan guardados en la base de datos
214     if form.validate_on_submit():
215         hashed_password=generate_password_hash(form.password.data)
216
217         conn = mysql.connector.connect(**db_config)
218         cursor = conn.cursor()
219         cursor.execute("SELECT * FROM usuario where username='{ }' or correo='{ }' ".format(form.username.data, form.correo.data))
220         data = cursor.fetchall()
221

```

La siguiente página es la de registrar Admin, se requiere tener una sesión activa y tener el rol de SuperAdmin. Utilizaremos el form de registrarAdmin.

Al hacer un request, la contraseña se convierte en un hash y se checa en la tabla usuario si el username o correo ya se han registrado.

```

221
222     #Si no se encuentra información en la tabla usuario, se procede a guardar la información, luego, se busca
223     #si el id o usuario estan en la tabla admin de la clínica correspondiente
224     if not data:
225         cursor.execute("INSERT INTO usuario (correo, username, contraseña, rol, clinica) VALUES (%s, %s, %s, %s, %s)", (form.correo.data,
226         cursor.execute("SELECT * FROM admin() where idAdmin='{ }' or usuarioAdmin='{ }' ".format(form.clinica.data, form.clinica.data)
227         data = cursor.fetchall()
228
229
230     #Si no se encuentra la información en la tabla admin de la clínica correspondiente, se procede a guardar la información
231     #y se hace commit para confirmar el guardado de información en la base de datos
232     if not data:
233         cursor.execute("INSERT INTO admin() VALUES (%s, %s, %s, %s, %s)".format(form.clinica.data),
234         (form.id.data, form.nombre.data, form.username.data, hashed_password, None, None))
235         conn.commit()
236         flash('El admin fue creado con éxito!','success')
237

```

Si no se encuentran en la tabla, se procede a guardar la información en la tabla usuario. Lo siguiente es buscar si el id o usuario han sido previamente guardados

en la tabla admin de la clínica correspondiente, si no se encuentran se procede a guardar la información en dicha tabla. Posteriormente se hace commit para confirmar el guardado de la información.

```
237
238
239 #Si en algunos de los casos anteriores, el id, correo o username se encuentran ya
240 #registrados en la base de datos Se le pide al usuario que cambie el dato correspondiente,
241 #asi mismo se hace un rollback para que no se quede información guardada a medias
242 else:
243     if data[0][0]==form.id.data:
244         flash("Ingrese otro id","danger")
245     else:
246         flash("Ingrese otro usuario","danger")
247         conn.rollback()
248         return render_template('registrarAdmin.html', form=form)
249
250 else:
251     if data[0][2]==form.username.data:
252         flash("Ingrese otro usuario","danger")
253     else:
254         flash("Ingrese otro correo","danger")
255         conn.rollback()
256         return render_template('registrarAdmin.html', form=form)
```

Si en alguno de los casos anteriores, el id, el correo o el username ya se encontraban registrados, se le avisa al usuario y se le pide que lo cambie.

```
256
257
258 #Cerramos la conexión a la base de datos
259 cursor.close()
260 conn.close()
261
262 #En caso de que el registro sea exitoso, se redirecciona a la página de inicio
263 return redirect(url_for('inicio'))
264
265 #Si no se envió ningún form, simplemente muestra la página de registrarAdmin
266 return render_template('registrarAdmin.html', form=form)
```

Se cierra la conexión con la base de datos y en caso de que el registro se éxito, se redirecciona a la pagina de inicio. Si no se envía ningún form, se muestra el template registrarAdmin.html.

```
267 #En esta función registraremos profesores, se requiere tener una sesión activa y tener el rol de SuperAdmin o de Admin
268 @app.route("/registrarProfesor", methods=['GET','POST'])
269 @login_required
270 @admin_permission.require(http_exception=403)
271 def registerProfesor():
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312 #En esta función registraremos alumnos, se requiere tener una sesión activa y tener el rol de SuperAdmin, de Admin o de Profesor
313 @app.route("/registrarAlumno", methods=['GET','POST'])
314 @login_required
315 @profesor_permission.require(http_exception=403)
316 def registerAlumno():
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

Las funciones de registrarProfesor, registraAlumno y registrarPacientes siguen el mismo formato que la de registrarAdmin.

[illegible]

Una de las distinciones que tiene registrarPaciente es que se necesitan llenar las tablas de las clínicas también, por lo que guardamos la información con el id del paciente y los demás valores como None. En el caso de odontología y fisioterapia se deberán actualizar los datos una vez que se consiga la información.

```

443
444 #Para el siguiente paso se crearan las carpetas que contendrán los archivos de cada paciente, primero se guarda como variable
445 #el id del paciente, se selecciona el form de la clínica correspondiente y se hace una iteración de los campos del form
446 #buscando aquellos que sean de tipo "FileField". Cuando se encuentra uno, se crea la carpeta con el formato
447 #SitioClinicas/static/Archivos/idPaciente/Clinica/Nombre del campo, el proceso se repite para las 3 clínicas
448 idPac=form.id.data
449 form=optometria()
450 for field in form:
451     if field.type == 'FileField':
452         if not os.path.exists(os.path.join("SitioClinicas/static/Archivos", idPac, "Optometria", field.name)):
453             os.makedirs(os.path.join("SitioClinicas/static/Archivos", idPac, "Optometria", field.name))
454
455 form=odontologia()
456 for field in form:
457     if field.type == 'FileField':
458         if not os.path.exists(os.path.join("SitioClinicas/static/Archivos", idPac, "Odontologia", field.name)):
459             os.makedirs(os.path.join("SitioClinicas/static/Archivos", idPac, "Odontologia", field.name))
460
461 form=fisioterapia()
462 for field in form:
463     if field.type == 'FileField':
464         if not os.path.exists(os.path.join("SitioClinicas/static/Archivos", idPac, "Fisioterapia", field.name)):
465             os.makedirs(os.path.join("SitioClinicas/static/Archivos", idPac, "Fisioterapia", field.name))
466 flash('El paciente fue registrado con éxito!', 'success')
467

```

Otra distinción en registrarPacientes es que se deben de crear las carpetas para los archivos que se deseen guardar, esto lo logramos primero guardando como variable el id del paciente, luego seleccionamos el form de la clínica y usamos un for para iterar los campos buscando aquellos que sean de tipo "FileField", al encontrarlos se crea la carpeta con la ruta: SitioClinicas/static/Archivos/idPaciente/Clínica/Nombre del campo, el proceso se repite para las 3 clínicas.

```

489 #En esta función consultaremos al personal, se requiere tener una sesión activa
490 @app.route("/consultas", methods=['GET','POST'])
491 @login_required
492 def consultas():
493     #Se utiliza el form consulta
494     form=consulta()
495
496     #Al hacer un request, nos conectamos a la base de datos y dependiendo del tipo de personal que busquemos, haremos una búsqueda en
497     #la tabla correspondiente con un join
498     if form.validate_on_submit():
499
500         conn = mysql.connector.connect(**db_config)
501         cur = conn.cursor()
502         if form.personal.data=="admin":
503             cur.execute("""SELECT per.id{}, per.nombre{}, per.usuario{}, us.correo from {}{} as per left join usuario as us on per.u
504
505         elif form.personal.data=="profesor":
506             cur.execute("""SELECT per.rfc{}, per.nombre{}, per.usuario{}, us.correo from {}{} as per left join usuario as us on per.
507
508         elif form.personal.data=="alumno":
509             cur.execute("""SELECT per.numeroCuenta{}, per.nombre{}, per.usuario{}, us.correo from {}{} as per left join usuario as u
510
511         data = cur.fetchall()
512         cur.close()
513         conn.close()

```

La función de consultas permite realizar búsquedas en el personal. Al realizar un request se realiza una búsqueda con un join en la base de datos dependiendo del tipo de personal.

```

514
515 #Obtenemos los nombres de las columnas de la tabla y combinamos los nombres de las columnas con los datos
516 column_names = [i[0] for i in cur.description]
517 data_with_columns = [dict(zip(column_names, row)) for row in data]
518
519 #Se mostrarán los resultados usando los parametros de data_with_columns, personal, clinica y rol
520 return render_template('Consultas.html', data_with_columns=data_with_columns, form=form, personal=form.personal.data, clinica=form
521 return render_template('Consultas.html', form=form)

```

Para finalizar, se obtienen los nombres de las columnas de las tablas y se combinan con la información obtenida. Se mostrarán los resultados usando los parámetros de data_with_columns, personal, clínica y rol.

```

523 #En esta función consultaremos a los pacientes, se requiere tener una sesión activa
524 @app.route("/consultasPacientes", methods=['GET','POST'])
525 @login_required
526 def Pacientes():
527     form=consultaPacientes()
528
529     #En este caso tenemos 2 botones, uno para buscar por id y otro para buscar por nombre
530     #Búsqueda a través de id
531     if 'submitIdPaciente' in request.form:
532         conn = mysql.connector.connect(**db_config)
533         cur = conn.cursor()
534         cur.execute("SELECT idPaciente, nombrePaciente FROM serviciosocial.paciente where idPaciente='{}'.format(form.idPaciente.data))
535         data = cur.fetchall()
536         cur.close()
537         conn.close()
538
539         column_names = [i[0] for i in cur.description]
540         data_with_columns = [dict(zip(column_names, row)) for row in data]
541
542         return render_template('ConsultasPacientes.html', data_with_columns=data_with_columns, form=form, )

```

La siguiente función nos permite hacer búsquedas de pacientes, en este caso tenemos 2 botones, el primero nos permite hacer la búsqueda mediante el id del paciente.

```

543
544 #Búsqueda a través del nombre
545 elif 'submitNombrePaciente' in request.form:
546     conn = mysql.connector.connect(**db_config)
547     cur = conn.cursor()
548     cur.execute("SELECT idPaciente, nombrePaciente FROM serviciosocial.paciente where nombrePaciente='{}'.format(form.nombrePaciente.
549     data = cur.fetchall()
550     cur.close()
551     conn.close()
552
553     column_names = [i[0] for i in cur.description]
554     data_with_columns = [dict(zip(column_names, row)) for row in data]
555
556     return render_template('ConsultasPacientes.html', data_with_columns=data_with_columns, form=form)
557
558 return render_template('ConsultasPacientes.html', form=form)

```

El segundo hace la búsqueda mediante el nombre del paciente.

```

559
560 #Esta función permite realizar modificaciones a mi perfil
561 @app.route("/perfil", methods=['GET','POST'])
562 @login_required
563 def perfiles():
564
565     #Dependiendo del rol que tengamos, se asignará el form
566     if current_user.rol=="Admin":
567         form=registrarAdmin()
568
569     elif current_user.rol=="Profesor":
570         form=registrarProfesor()
571
572     elif current_user.rol=="Alumno":
573         form=registrarAlumno()
574
575     #Se eliminan del form el campo de password y confirm_password para evitar la validación obligatoria de estos
576     #Se podrán cambiar en otro lugar
577     del form.password
578     del form.confirm_password
579

```

La siguiente función es la de perfil, esta permite realizar modificaciones al perfil del usuario. Dependiendo del rol que tengamos se asignará el form. Se eliminan del form password y confirm_password para evitar la validación obligatoria.

```

580
581 #Al realizar un request, se guardan los cambios en las tablas correspondientes dependiendo del rol que tengamos
582 if form.validate_on_submit():
583     conn = mysql.connector.connect(**db_config)
584     cur = conn.cursor()
585     if current_user.rol=="Admin":
586         cur.execute("UPDATE admin{} set idAdmin()='{}', nombreAdmin()='{}', usuarioAdmin()='{}' where usuarioAdmin()='{}' "
587             .format(current_user.clinica, current_user.clinica, form.id.data, current_user.clinica, form.nombre.data, current_user.clinica))
588         cur.execute("UPDATE usuario set username()='{}', correo()='{}' where username()='{}' "
589             .format(form.username.data, form.correo.data, current_user.username))
590
591     elif current_user.rol=="Profesor":
592         cur.execute("UPDATE profesor{} SET rfcProfesor()='{}', nombreProfesor()='{}', apellidosProfesor()='{}', usuarioProfesor()='{}' "
593             .format(current_user.clinica, current_user.clinica, form.rfc.data, current_user.clinica, form.nombre.data, current_user.clinica))
594         cur.execute("UPDATE usuario set username()='{}', correo()='{}' where username()='{}' "
595             .format(form.username.data, form.correo.data, current_user.username))
596
597     elif current_user.rol=="Alumno":
598         cur.execute("UPDATE alumno{} SET numeroCuentaAlumno()='{}', nombreAlumno()='{}', apellidosAlumno()='{}', año='{}', asignaturaAlumno()='{}' "
599             .format(current_user.clinica, current_user.clinica, form.numeroCuenta.data, current_user.clinica, form.nombre.data, current_user.clinica, form.asignatura.data))
600         cur.execute("UPDATE usuario set username()='{}', correo()='{}' where username()='{}' "
601             .format(form.username.data, form.correo.data, current_user.username))
602
603     conn.commit()
604     cur.close()
605     conn.close()
606     flash("Información guardada", "success")
607     #Se hace el commit, se cierra la conexión y se redirecciona a la página de inicio
608     return redirect(url_for('inicio'))

```

Al realizar un request, se hacen los cambios en la base de datos dependiendo del rol que tengamos. Se hace el commit, se cierra la conexión y se redirecciona a la página de inicio.

```

609 #Al ser una página donde se muestra información, esta debe de ser buscada al momento de cargar la página
610 #Igual que antes, la búsqueda se realiza dependiendo del rol que tengamos
611 conn = mysql.connector.connect(**db_config)
612 cur = conn.cursor()
613 if current_user.rol=="Admin":
614     form=registrarAdmin(clinica=current_user.clinica)
615     cur.execute("""SELECT adm.idAdmin(), adm.nombreAdmin(), adm.usuarioAdmin(), us.correo from admin() as adm left join usuario as us
616         .format(current_user.clinica, current_user.clinica, current_user.clinica, current_user.clinica, current_
617
618 elif current_user.rol=="Profesor":
619     form=registrarProfesor(clinica=current_user.clinica)
620     cur.execute("""SELECT prof.rfcProfesor(), prof.nombreProfesor(), prof.apellidosProfesor(), prof.usuarioProfesor(), us.correo from
621         .format(current_user.clinica, current_user.clinica, current_user.clinica, current_user.clinica, current_user.clinica, current_
622
623 elif current_user.rol=="Alumno":
624     form=registrarAlumno(clinica=current_user.clinica)
625     cur.execute("""SELECT alum.numeroCuentaAlumno(), alum.nombreAlumno(), alum.apellidosAlumno(), alum.año, alum.asignaturaAlumno(), a
626         .format(current_user.clinica, current_user.clinica, current_user.clinica, current_user.clinica, current_user.clinica, current_
627
628 data = cur.fetchall()
629 cur.close()
630 conn.close()
631
632 #Se muestran los resultados con datos y rol como parametros
633 return render_template('infoPerfil.html', data=data, form=form, rol=current_user.rol)
634

```

Al ser una página donde se muestra información, se debe realizar la búsqueda al cargar la página. Dependiendo del rol que tengamos se hace la búsqueda con un join en las tablas correspondientes. La información se muestra con data como parámetro.

```

636 #En esta función podemos cambiar la contraseña, se requiere una sesión activa
637 @app.route("/perfil/cambiarContraseña", methods=['GET','POST'])
638 @login_required
639 def cambiarContraseña():
640     form=CambiarForm()
641
642     #Al recibir un request, la contraseña se convierte en un hash y se guarda en l base de datos
643     if form.validate_on_submit():
644         hashed_password=generate_password_hash(form.password.data)
645
646         conn = mysql.connector.connect(**db_config)
647         cursor = conn.cursor()
648
649         cursor.execute("UPDATE usuario set contraseña='{}' where id='{}' ".format(hashed_password, current_user.id))
650         conn.commit()
651
652         # Cierra la conexión a la base de datos
653         cursor.close()
654         conn.close()
655         flash('La contraseña ha sido cambiada', 'success')
656         return redirect(url_for('inicio'))
657     return render_template('cambiarContraseña.html', form=form)
658

```

La siguiente función nos permite cambiar la contraseña, al recibir un request, la contraseña se cambia por un hash por seguridad, se busca el usuario en la base de datos y se cambia la contraseña.

```

659
660 #En esta función podemos ver la información de los pacientes, se requiere una sesión activa
661 #Al presionar el botón de "consulta" en la página de consulta de pacientes, nos redirecciona a esta página
662 #Tomando el id del paciente como parametro
663 @app.route("/paciente/<id>", methods=['GET','POST'])
664 @login_required
665 def infoPaciente(id):
666     #Utilizamos el id del paciente como variable
667     idPac=id
668     form=registrarPaciente()
669
670     #Al hacer un request, podemos actualizar la información del paciente
671     if form.validate_on_submit():
672         conn = mysql.connector.connect(**db_config)
673         cur = conn.cursor()
674         cur.execute("""UPDATE paciente SET fechaRemision='{0}', fechaAtención='{0}', sexo='{0}', edad='{0}', nombrePaciente='{0}', apellidoP
675             extensionTelefono='{0}', telefonoOficina='{0}', celular='{0}', email='{0}', calle='{0}', numeroExterior='{0}', numeroInte
676             nacionalidad='{0}', tipoSangre='{0}', clasificacion='{0}', alertaAdministrativa='{0}', numeroSeguro='{0}', nombreContact
677             (form.fechaRemision.data, form.fechaAtención.data, form.sexo.data, form.edad.data, form.nombre.data, form.apellidoP
678             form.telefonoCasa.data, form.extensionTelefono.data, form.telefonoOficina.data, form.celular.data, form.email.data,
679             form.codigoPostal.data, form.estado.data, form.pais.data, form.origen.data, form.estadoCivil.data, form.ocupacion.d
680             form.alertaAdministrativa.data, form.numeroSeguro.data, form.nombreContactoEmergencia.data, form.telefonoContactoEm
681         conn.commit()
682         flash('Las modificaciones fueron realizadas con éxito!','success')
683
684     # Cierra la conexión a la base de datos
685     cur.close()
686     conn.close()

```

La siguiente función nos permite visualizar la información del paciente, esta se accede mediante el botón “Consultar” en la página de consulta de pacientes utilizando el id del paciente como parámetro.

Dentro de la función tomamos el id del paciente como variable y al recibir un request, podemos actualizar la información del paciente.

```

688 #Al cargar la página queremos mostrar la información del paciente,
689 #por lo que hacemos una búsqueda de la información del paciente mediante su id
690 conn = mysql.connector.connect(**db_config)
691 cur = conn.cursor()
692 cur.execute("SELECT * FROM serviciosocial.paciente where idPaciente='{0}'".format(idPac))
693 data = cur.fetchall()
694 cur.close()
695 conn.close()
696
697 #Debido a que estamos usando Form Flask, los campos de tipo "SelectField" deben definirse al seleccionar el form
698 form=registrarPaciente(sexo=data[0][4], estado=data[0][23], pais=data[0][24], estadoCivil=data[0][26], ocupacion=data[0][27], tipoSang
699 return render_template('infoPaciente.html', data=data, idPac=idPac, form=form)
700

```

Al cargar la página queremos mostrar la información, por lo que realizamos una búsqueda de la información del paciente. Debido a que estamos usando Flask Form, los campos de tipo “SelectField” deben definirse al seleccionar el form.

```

701 #En esta función podemos ver la información del paciente de la clínica de odontología
702 @app.route("/paciente/<id>/odontologia", methods=['GET','POST'])
703 @login_required
704 def infoOdontologia(id):
705     idPac=id
706     form=odontologia()
707
708     #Al recibir un request, debemos guardar los archivos en las carpetas previamente creadas
709     #Iteramos los campos del form buscando los que sean de tipo "FileField" y si el campo tiene un archivo,
710     #se procede a guardarlo en su carpeta correspondiente con el formato:
711     #SitioClinicas/static/Archivos/idPaciente/Clinica/Nombre del campo/Nombre del archivo
712     if form.validate_on_submit():
713         for field in form:
714             if field.type == 'FileField':
715                 f = field.data
716                 if f != None:
717                     filename = secure_filename(f.filename)
718                     f.save(os.path.join("SitioClinicas/static/Archivos", idPac, "Odontologia", field.name, filename))
719
720     #Posteriormente se guarda la información en la base datos, en el caso de la clínica de optometría, se deberán actualizar estos
721     #datos al recibir la información de parte de la clínica
722     conn = mysql.connector.connect(**db_config)
723     cursor = conn.cursor()
724
725     cursor.execute("""UPDATE odontologia Set octDiscoOptico=CONCAT("SitioClinicas/static/Archivos/",'{}','/Odontologia/octDiscoOptico'
726     anguloIridocorneal=CONCAT("SitioClinicas/static/Archivos/",'{}','/Odontologia/anguloIridocorneal'), proximaVisita=
727     .format(idPac, idPac, idPac, idPac, idPac, form.proximaVisita.data, idPac))
728
729     conn.commit()
730     cursor.close()
731     conn.close()

```

La siguiente función nos permite visualizar y modificar la información de un paciente en la clínica de odontología. Al recibir un request, debemos guardar los archivos en sus carpetas correspondientes, por lo que iteramos los campos del form buscando los que sean de tipo “FileField”, si tienen información, se procederá a guardarlo con el siguiente formato: SitioClinicas/static/Archivos/idPaciente/Clinica/Nombre del campo/Nombre del archivo. Posteriormente se guarda la información en la base de datos.

Las sentencias SQL de las clínicas de odontología y de fisioterapia deberán ser modificadas al recibir la información que ellas requieran.

```

732
733     #Al cargar la página queremos mostrar la información del paciente,
734     #por lo que hacemos una búsqueda de la información del paciente mediante su id
735     conn = mysql.connector.connect(**db_config)
736     cur = conn.cursor()
737     cur.execute("SELECT * FROM serviciosocial.odontologia where idPaciente='{}'".format(idPac))
738     data = cur.fetchall()
739     cur.close()
740     conn.close()
741
742     return render_template('odontologia.html', form=form, idPac=idPac, data=data)

```

Al cargar la página queremos mostrar la información del paciente, por lo que hacemos una búsqueda de la información del paciente mediante su id

Las funciones para Fisioterapia y Optometría siguen el mismo formato, lo único que cambia son las rutas de las carpetas y las tablas donde se hacen las búsquedas.


```

743
744 #En esta función podemos visualizar los archivos guardados del paciente de la clínica de odontología
745 @app.route("/paciente/<id>/odontologia/archivos", methods=['GET', 'POST'])
746 @login_required
747 def archivosOdontologia(id):
748     form=odontologia()
749     idPac=id
750
751     #Tomamos como variable la primera parte fija de la ruta donde se encuentran los archivos
752     #Primero iteramos los campos del form, después iteramos las carpetas del paciente en la clínica
753     #Si el nombre de la carpeta y el nombre el campo coinciden, se agregan a una lista
754     #el nombre de la carpeta, el label del campo y los archivos dentro de las carpetas
755     dir_path = "SitioClinicas/static/Archivos"
756     res = []
757
758     for field in form:
759         for path in os.listdir(os.path.join(dir_path, idPac, "Odontologia")):#Viendo el nombre de las carpetas
760             if field.name==path:
761                 res2=[path, field.label, os.listdir(os.path.join(dir_path, idPac, "Odontologia", path))]
762                 res.append(res2)
763
764 #Se muestra el template correspondiente tomando como parametros el id del paciente y la lista antes mencionada
765 return render_template('archivosOdontologia.html', idPac=idPac, res=res)

```

La siguiente función nos permite visualizar los archivos del paciente de la clínica de odontología.

Tomamos como variable la primera parte fija de la ruta donde se encuentran los archivos. Primero iteramos los campos del form, después iteramos las carpetas del paciente en la clínica, si el nombre de la carpeta y el nombre el campo coinciden, se agregan a una lista el nombre de la carpeta, el label del campo y los archivos dentro de las carpetas.

Finalmente se muestra el template correspondiente tomando como parámetros el id del paciente y la lista antes mencionada.

Las funciones para Fisioterapia y Optometría siguen el mismo formato, lo único que cambia son las rutas de las carpetas.

```

902
903 #Esta función a traves de un JQuery en el html que permite la visualización de los archivos
904 #La función permite la eliminación de los archivos
905 @app.route('/background_process_test')
906 def background_process_test():
907     #Se recibe como variable la parte de la ruta que incluye el id del paciente, clínica y el nombre del archivo
908     #En caso de que el archivo sea eliminado se informa al usuario
909     try:
910         prueba = request.args.get('prueba', 0)
911         dir_path = "SitioClinicas/static/Archivos/"+prueba
912         os.remove(dir_path)
913         flash ("Archivo eliminado", 'success')
914     except:
915         flash("El archivo ya fue eliminado", "succes")
916     return ("nothing")

```

La siguiente función es llamada a través de un JQuery en el html que permite la visualización de los archivos. Esta función permite la eliminación de los archivos. Recibe como variable la parte de la ruta que incluye el id del paciente, la clínica y el nombre del archivo. En caso de que la eliminación sea exitosa, se le informa al usuario.


```

917
918 #Esta función es llamada traves de un JQuery en el html que permite la consulta del personal
919 #La función permite la eliminación del personal
920 @app.route('/background_process_test2')
921 def background_process_test2():
922
923     #Se recibe como variable la el id, clínica y rol del usuario
924     #Se busca y se elimina al usuario de la base de datos
925     #En caso de que el archivo sea eliminado se informa al usuario
926     try:
927         idUsuario = request.args.get('id', 0)
928         clinica = request.args.get('clinica', 0)
929         personal = request.args.get('personal', 0)
930
931         conn = mysql.connector.connect(**db_config)
932         cur = conn.cursor()
933         cur.execute("SELECT usuario({}) from {} where id({})='{}'.format(personal,clinica,personal,clinica,personal,clinica,idUsuario))
934         data = cur.fetchall()
935         cur.execute("DELETE from usuario where username='{}'.format(data[0])")
936         cur.execute("DELETE from {} where usuario({})='{}'.format(personal,clinica,personal,clinica,data[0])")
937         conn.commit()
938         cur.close()
939         conn.close()
940         flash("Usuario eliminado", 'success')
941
942     except:
943         flash("El usuario ya fue eliminado", "succes")
944     return ("nothing")
945

```

La siguiente función también es llamada a través de un JQuery en el html que permite la consulta del personal. Esta función permite la eliminación del personal. Recibe como variable el id, clínica y rol del usuario. Se busca y se elimina al usuario de la base de datos. En caso de que la eliminación sea exitosa, se le informa al usuario.

```

946 #La función de logout nos permite cerrar sesión
947 @app.route('/logout')
948 def logout():
949
950     #Usamos la función logout_user para cerrar sesión, removemos la identidad y redireccionamos a la página de inicio de sesión
951     logout_user()
952     for key in ('identity.name', 'identity.auth_type'): session.pop(key, None)
953     identity_changed.send(current_app._get_current_object(), identity=AnonymousIdentity())
954
955     return redirect(url_for('hello_world'))
956

```

La función de logout nos permite cerrar sesión. Usamos la función `logout_user` de Flask Login para cerrar sesión, removemos la identidad y redireccionamos a la página de inicio de sesión.

```

957 #Esta función nos permite obtener un link para cambiar nuestra contraseña en caso de olvidarla
958 @app.route('/recuperarContrasena', methods=['GET','POST'])
959 def recuperar():
960     form=RecuperarForm()
961
962     #Al recibir un request se busca en la base datos si el correo introducido existe
963     if form.validate_on_submit():
964         conn = mysql.connector.connect(**db_config)
965         cur = conn.cursor()
966         cur.execute("SELECT id,correo FROM serviciosocial.usuario where correo='{}'.format(form.correo.data))
967         data = cur.fetchall()
968         cur.close()
969         conn.close()
970         column_names = [i[0] for i in cur.description]
971         data_with_columns = [dict(zip(column_names, row)) for row in data]
972

```

Esta función nos permite obtener un link para recuperar la contraseña en caso de olvidarla. Al recibir un request, se busca si el correo ingresado existe en la base de datos.

```

972
973 #En caso de que si exista, se crea un token que contendrá el id del usuario asociado al correo
974 #Y se mandará un correo con el link que incluye el token
975 if data!=[]:
976     serial=Serializer(app.config['SECRET_KEY'],expires_in=300)
977     token=serial.dumps({'user_id':(data_with_columns[0])['id']}).decode('utf-8')
978     msg=Message('Cambio de contraseña', recipients=[(data_with_columns[0])['correo']],sender='noreply@correo.com')
979     msg.body=f''Para cambiar tu contraseña presiona el link:
980
981     {url_for('recuperar_token',token=token,_external=True)}
982
983     ...
984
985     mail.send(msg)
986     flash('Correo enviado', 'success')
987     return redirect(url_for('hello_world'))
988 else:
989     flash('Correo no encontrado', 'danger')
990 return render_template('recuperarContraseña.html', form=form)
991

```

En caso de que el correo exista, se crea un token que incluye el id del usuario y que expira en 5 minutos. Posteriormente se manda un correo que incluye un link con el token. En caso de que el proceso sea exitoso, se le informa al usuario y lo redirecciona a la pagina de inicio de sesión.

```

992 #Esta función es una continuación de la función anterior ya que es la
993 #correspondiente al token creado, permite cambiar la contraseña
994 @app.route('/recuperarContraseña/<token>', methods=['GET','POST'])
995 def recuperar_token(token):
996
997     #Se inicializa el Serializer para poder leer el token
998     serial2=Serializer(app.config['SECRET_KEY'])
999     form=CambiarForm()
1000
1001     #Se lee el token y se obtiene el id del usuario, en caso contrario se le informa al usuario
1002     try:
1003         user_id=serial2.loads(token)['user_id']
1004     except:
1005         flash('No se encuentra el usuario o el link expiró', 'danger')
1006         return redirect(url_for('recuperar'))
1007

```

La siguiente función es una continuación de la anterior ya que corresponde al token creado anteriormente. En esta función podemos cambiar la contraseña. Primero iniciamos el serializer para poder leer el token, obtenemos el id del token. En caso contrario se le informa al usuario.

```

1007
1008 #Al recibir un request, nos conectamos a la base de datos y se cambia la contraseña del usuario
1009 if form.validate_on_submit():
1010     hashed_password=generate_password_hash(form.password.data)
1011
1012     conn = mysql.connector.connect(**db_config)
1013     cursor = conn.cursor()
1014
1015     cursor.execute("UPDATE usuario set contraseña='{}' where id='{}' ".format(hashed_password,user_id))
1016
1017     conn.commit()
1018     cursor.close()
1019     conn.close()
1020     flash('La contraseña se ha cambiado, por favor inicia sesion', 'success')
1021     return redirect(url_for('hello_world'))
1022 return render_template('cambiarContraseña.html', form=form)
1023

```

Al recibir un request, accedemos a la base de datos para cambiar la contraseña del usuario y se le informa al usuario.

```

1023
1024 #Esta función maneja el error 403 que aparece cuando alguien
1025 #intenta acceder a alguna página sin los permisos necesarios
1026 @app.errorhandler(403)
1027 def page_not_found(e):
1028     flash("No tienes permiso para acceder a este recurso","danger")
1029     return redirect(url_for('inicio'))

```

La última función es la encargada de manejar el error 403 que aparece cuando alguien intenta acceder a una página sin los permisos necesarios. Mostrará un mensaje indicando al usuario y redireccionará a la página de inicio.

agregarSuperAdmin.py

```

agregarSuperAdmin.py  run.py  Prueba.py
1  from werkzeug.security import generate_password_hash
2  import mysql.connector
3
4  # Configura los detalles de la conexión a la base de datos
5  db_config = {
6      'user': 'root',
7      'password': 'contraseña',
8      'host': 'localhost',
9      'database': 'serviciosocial',
10 }
11
12 def agregar_usuario(id, usuariosuperAdmin, password, rol):
13     # Genera el hash de la contraseña utilizando werkzeug
14     password_hashsuperAdmin = generate_password_hash(password)
15
16     # Conecta a la base de datos
17     conn = mysql.connector.connect(**db_config)
18     cursor = conn.cursor()
19
20     # Inserta el nuevo usuario en la base de datos
21     cursor.execute('INSERT INTO usuario (username, contraseña, rol) VALUES (%s, %s, %s)', (usuariosuperAdmin, password_hashsuperAdmin, rol))
22     conn.commit()
23     cursor.execute('INSERT INTO superAdmin VALUES (%s, %s, %s, %s)', (id, usuariosuperAdmin, password_hashsuperAdmin, None))
24     conn.commit()
25     # Cierra la conexión a la base de datos
26     cursor.close()
27     conn.close()
28
29     print('El usuario se ha agregado exitosamente.')
30
31 # Ejemplo de uso
32 #nuevo_id = input('Ingrese el id del usuario: ')
33 #nuevo_username = input('Ingrese el nombre de usuario: ')
34 #nuevo_password = input('Ingrese la contraseña: ')
35 #nuevo_rol = input('Ingrese el rol: ')
36
37 nuevo_id = "1"
38 nuevo_username = "SuperAdmin1"
39 nuevo_password = "123"
40 nuevo_rol = "SuperAdmin"
41
42 agregar_usuario(nuevo_id, nuevo_username, nuevo_password, nuevo_rol)

```

Este código es para agregar SuperAdmins quien tiene permiso para agregar admins y eliminar usuarios. Se ejecuta por separado del código del sitio web, por esa razón hay que definir los valores para la conexión de la base de datos. Se crea una función para insertar el SuperAdmin en la base de datos. Las variables se pueden llenar con un input o directamente el código. Finalmente se llama la función. En caso de que el registro sea exitoso, se le avisa mediante un print en consola.