```python
"""Module to execute user operations for Administrator Role."""

import secrets
import string
from argon2 import PasswordHasher
import psycopg2
from psycopg2 import sql
import dbconnection as dbc
import eventlog as log
import notification

RED = '\033[91m' # Erorr Messages
GREEN = '\033[92m' # Success Messages
YELLOW = '\033[93m' # Notices to User

def register_new_user(first_name:str,last_name:str,dob:str,email:str,role:int,admin_id:int) -> bool: #pylint: disable=too-many-arguments, disable=too-many-locals
    '''
    Function to sign up a new user. Takes user information as arg to sign up accordingly.
    Takes Admin's id as argument to create log in the eventlog database.
    If sign-up is successful, triggers notification email.
    '''
    conn = dbc.establish_connection('authentication')
    cursor = conn.cursor() # Connect Cursor to Authentication DB
    username = generate_username(first_name, last_name)
    password = generate_password(12)
    clear_pswd = password[0]
    hash_pswd = password[1]
    psql = """
          INSERT INTO users
          (first_name, last_name, dob, user_role, username, password, status, email)
          VALUES
          (%(first_name)s,%(last_name)s,%(dob)s,%(role)s,%(uname)s,%(pw)s,%(stat)s,%(email)s)
          RETURNING id
          """
    val = {
        'first_name':first_name,
        'last_name':last_name,
        'dob':dob,
        'role':role,
        'uname':username,
        'pw':hash_pswd,
        'stat':1,
        'email':email
        }
    try:
        cursor.execute(psql, val)
        conn.commit()
    except psycopg2.OperationalError as error:
        print(RED + 'Issue with registering new user on database. Error:', error)
        print(YELLOW + 'Error TYPE:', type(error))
        return False
    except psycopg2.errors.DatetimeFieldOverflow: #pylint: disable=no-member
        print(RED + 'Issue with the given Date of Birth. Please check your input and try again.')
        return False
    else:
        created_user_id = cursor.fetchone()[0]
        log.admin_log('Create User', admin_id, created_user_id) # logging event in logs
        print(GREEN + f'User successfully created. Sending Registration email to {email}...')
    sent_email = notification.registration_email(first_name, email, username, clear_pswd)
    if sent_email:
        print(GREEN + 'Email sent successfully!')
    else:
```

```
62              print(RED + 'Email could not be sent. Please ensure that the SMTP is reachable.')
63          return True
64
65
66      def modify_user(uid:int, attribute:str, new_value:str, admin_id:int) -> bool:
67          '''
68          Function to modify an existing user. Takes the user id as input to execute upon.
69          The attribute sets the field to be modified, the new_value denotes the value after
            modification.
70          '''
71          conn = dbc.establish_connection('authentication')
72          cursor = conn.cursor() # Connect Cursor to Authentication DB
73
74          stmt = sql.SQL("SELECT {attribute} FROM users WHERE id = {uid}").format(
75                  attribute = sql.Identifier(attribute),
76                  uid = sql.Literal(uid),
77          )
78          cursor.execute(stmt)
79
80          curr_val = cursor.fetchall()[0][0]
81          if attribute == 'user_role':
82              new_value = int(new_value) # change new value to int type if the user role is
                being changed
83
84          stmt = sql.SQL("UPDATE users SET {attribute} = {value} WHERE id = {uid}").format(
85                  attribute = sql.Identifier(attribute),
86                  uid = sql.Literal(uid),
87                  value = sql.Literal(new_value),
88          )
89
90          try:
91              cursor.execute(stmt)
92              conn.commit()
93          except psycopg2.OperationalError as error:
94              print(RED + 'Issue with modifying user on database. Error:', error)
95              print(YELLOW + 'Error TYPE:', type(error))
96              return False
97          except psycopg2.errors.DatetimeFieldOverflow: #pylint: disable=no-member
98              print(RED + 'Issue with the given Date of Birth. Please check your input and
                try again.')
99              return False
100         log.admin_log(
101             'Edit User',
102             admin_id,
103             uid,
104             modified=attribute,
105             old_val=str(curr_val),
106             new_val=str(new_value)
107             ) # log Edit User event
108         return True
109
110
111     def unlock_user(uid:int, admin_id:int) -> bool:
112         '''
113         Function to unlock an already locked user. Input: admin's id and the id of the user
            to unlock.
114         If unlock was successful, will return bool 'True'. If there was an error, returns
            bool 'False'.
115         '''
116         conn = dbc.establish_connection('authentication')
117         cursor = conn.cursor() # Connect Cursor to Authentication DB
118         psql = "UPDATE users SET status=1 WHERE id=%(uid)s"
119         val = {'uid':uid}
120         try:
121             cursor.execute(psql,val)
122             conn.commit()
123         except psycopg2.OperationalError as error:
```

```
124             print(RED + 'Issue with unlocking user on database. Error:', error)
125             print(YELLOW + 'Error TYPE:', type(error))
126             return False
127         log.admin_log('Unlock User', admin_id, uid, modified='status', old_val='3',
            new_val='1')
128         return True
129
130
131     def lock_user(uid:int) -> bool:
132         '''
133         Function to lock a user if there are more than three failed login attempts. Input:
            User's ID.
134         If lock was successful, will return bool 'True'. If there unsuccessful, returns
            bool 'False'.
135         '''
136         conn = dbc.establish_connection('authentication')
137         cursor = conn.cursor() # Connect Cursor to Authentication DB
138         psql = "UPDATE users SET status=3 WHERE id=%(uid)s;"
139         val = {'uid':uid}
140         try:
141             cursor.execute(psql,val)
142             conn.commit()
143         except psycopg2.OperationalError as error:
144             print(RED + 'Issue with locking user on database. Error:', error)
145             print(YELLOW + 'Error TYPE:', type(error))
146             return False
147         log.auth_log("Account Locked", uid) # log locked account event
148         return True
149
150
151     def deactivate_user(uid:int, admin_id:int, curr_status:int) -> bool:
152         '''
153         Function to deactivate a user if the access is no longer needed. Input: User's ID.
154         If deactivation was successful, will return bool 'True'. If not, returns bool
            'False'.
155         '''
156         conn = dbc.establish_connection('authentication')
157         cursor = conn.cursor() # Connect Cursor to Authentication DB
158         psql = "UPDATE users SET status=2 WHERE id=%(uid)s;"
159         val = {'uid':uid}
160         try:
161             cursor.execute(psql,val)
162             conn.commit()
163         except psycopg2.OperationalError as error:
164             print(RED + 'Issue with deactivating user on database. Error:', error)
165             print(YELLOW + 'Error TYPE:', type(error))
166             return False
167         log.admin_log(
168             'Deactivate User',
169             admin_id, uid,
170             modified='status',
171             old_val=str(curr_status),
172             new_val='2'
173             ) # log deactivation event
174         return True
175
176
177     def fetch_user_info(uid=None, email=None, username=None) -> tuple: #pylint:
        disable=too-many-return-statements
178         '''
179         Queries user information based on the given email, username or uid.
180         Returns a tuple of user id, first name, last name, email, dob and status if user
            was found.
181         Returns None if user was not found.
182         '''
183         cursor = dbc.establish_connection('authentication').cursor() # Connect Cursor to
            Auth DB
```

```
184         if uid is not None:
185             psql = "SELECT * FROM users WHERE id=%(val)s"
186             val = {'val':uid}
187             try:
188                 cursor.execute(psql,val)
189                 result = cursor.fetchall()[0]
190             except IndexError:
191                 return None
192             return (result[0], result[1], result[2], result[9], result[3], result[8])
193         if email is not None:
194             psql = "SELECT * FROM users WHERE email=%(val)s"
195             val = {'val':email}
196             try:
197                 cursor.execute(psql,val)
198                 result = cursor.fetchall()[0]
199             except IndexError:
200                 return None
201             return (result[0], result[1], result[2], result[9], result[3], result[8])
202         if username is not None:
203             psql = "SELECT * FROM users WHERE username=%(val)s"
204             val = {'val':username}
205             try:
206                 cursor.execute(psql,val)
207                 result = cursor.fetchall()[0]
208             except IndexError:
209                 return None
210             return (result[0], result[1], result[2], result[9], result[3], result[8])
211         return None
212 
213 
214     def fetch_all_authorities() -> list:
215         '''
216         Query to fetch all users in the system with the 'Authority' role.
217         Returns a list of tuples containing pairs of emails and first_names
218         to be used by the source notification email.
219         '''
220         cursor = dbc.establish_connection('authentication').cursor() # Connect Cursor to
            Auth DB
221         psql = "SELECT * FROM users WHERE user_role=3"
222         cursor.execute(psql)
223         result = cursor.fetchall()
224         output = []
225         for item in result:
226             output.append((item[9], item[1]))
227         return output
228 
229 
230     def generate_password(length:int) -> tuple:
231         '''
232         Function to generate a random, secure password with the given length.
233         Returns the clear password, as well as the argon2 hash of the password.
234         '''
235         password = ''.join((secrets.choice(string.ascii_letters + string.digits +
            string.punctuation)
236                             for i in range(length)))
237         a2_ph = PasswordHasher() # set password hasher for Argon2
238         hashed = a2_ph.hash(password)
239         return (password, hashed)
240 
241 
242     def generate_username(first_name, last_name) -> str:
243         '''
244         Function to return a valid username for the given first and last name.
245         If combination is taken already, will add a running number to the end of the
            username.
246         '''
247         user_comb = first_name[0] + '.' + last_name
```

```
248         user_comb = user_comb.lower()
249         if not username_exists(user_comb):
250             return user_comb
251         running_number = 1
252         while True:
253             if not username_exists(user_comb+str(running_number)):
254                 return user_comb+str(running_number)
255             running_number += 1
256 
257 
258     def username_exists(username:str) -> bool:
259         '''
260         Checks the users database if a given username exists in the DB already.
261         If not, returns bool 'False', if it exists returns bool 'True'.
262         '''
263         cursor = dbc.establish_connection('authentication').cursor() # Connect Cursor to
            Auth DB
264         psql = "SELECT count(*) FROM users WHERE username=%(val)s"
265         val = {'val':username}
266         cursor.execute(psql,val)
267         result = cursor.fetchone()[0]
268         if result == 0:
269             return False
270         return True
271 
272     """This module handles the login and password hashing functionality."""
273 
274     from datetime import datetime
275     import sys
276     import argon2 # Argon2 lib to hash password
277     import psycopg2
278     import dbconnection as dbc
279     import eventlog as log
280 
281     WHITE = '\033[97m' # User Input
282     RED = '\033[91m' # Error Messages
283 
284     # set Argon2 password hasher object
285     ph = argon2.PasswordHasher()
286 
287     def hash_pswd(password:str) -> str:
288         """
289         Uses Argon2 to hash password and returns hash as string.
290         Takes clear text password string as input.
291         """
292         hashed = ph.hash(password)
293         return hashed
294 
295 
296     def existing_user(user:str, password:str) -> tuple: #pylint:
        disable=inconsistent-return-statements
297         '''
298         Function to authenticate existing user against the database.
299         Takes username and clear password as input.
300         If successful, returns authenticated user as tuple: (user id, first name, user role).
301         If username not found or user is locked or deactivated, returns None.
302         If password was incorrect, returns False.
303         '''
304         cursor = dbc.establish_connection('authentication').cursor()
305         sql = 'SELECT id, first_name, user_role, password, status FROM users WHERE username
            = %(val)s'
306         val = {'val':user}
307         try:
308             cursor.execute(sql,val)
309             result = cursor.fetchall()[0]
310         except psycopg2.OperationalError as error:
311             print(RED+"Encountered an issue with the database. Error:", error)
```

```
312             print(WHITE, end='')
313             return None
314         except IndexError:
315             return None
316         user_status = result[4]
317         if user_status==2:
318             log.auth_log("Failed Login: Deactivated User", result[0])
319             print('This user is deactivated.', end='')
320             print('Please contact the system administrator team for further information.')
321             print(WHITE)
322             sys.exit()
323         elif user_status==3:
324             log.auth_log("Failed Login: Locked User", result[0])
325             print(RED+'This user is currently locked. ', end='')
326             print('Please contact the system administrator team for further information.')
327             print(WHITE)
328             sys.exit()
329         try:
330             match = ph.verify(result[3], password)
331         except argon2.exceptions.VerifyMismatchError:
332             return False
333         if match:
334             return (result[0], result[1], result[2])
335
336
337     def update_last_login(uid:int) -> bool:
338         '''
339         Function to update a user's last login value with the current datetime stamp.
340         Returns True if successful and False if not.
341         '''
342         conn = dbc.establish_connection('authentication')
343         cursor = conn.cursor()
344         now = datetime.now()
345         sql = 'UPDATE users SET last_login = %(now)s WHERE id = %(uid)s'
346         val = {'now':now, 'uid':uid}
347         try:
348             cursor.execute(sql,val)
349             conn.commit()
350         except psycopg2.OperationalError:
351             return False
352         return True
353
354
355     def fetch_last_login(uid:int) -> str:
356         '''
357         Function to fetch a user's last login date.
358         Returns date string if existent or None if empty.
359         '''
360         cursor = dbc.establish_connection('authentication').cursor()
361         sql = 'SELECT last_login FROM users WHERE id=%(uid)s'
362         val = {'uid':uid}
363         try:
364             cursor.execute(sql,val)
365             result = cursor.fetchall()[0][0]
366         except IndexError:
367             return None
368         return result
369
370     """This module establishes a connection to the PostgreSQL DB."""
371
372     from cryptography.fernet import Fernet # lib to decrypt Postgresql credentials from
    binary file
373     import psycopg2 # Postgresql connector library
374
375     RED = '\033[91m' # Erorr Messages
376     YELLOW = '\033[93m' # Notices to User
377
```

```
def retrieve_key():
    '''Function to retrieve the Fernet Encryption Key from the config folder.'''
    # Try retrieving the Fernet encryption key from bin file
    try:
        key_file = open("config/key.bin", "rb")
        retrieved_key = key_file.read()
        key_file.close()
    except OSError:
        print(RED + "Error retrieving key.")
        return None
    return retrieved_key

# Try retrieving the Postgresql credentials from bin file
try:
    loginFRetrieve = open("config/credentials.bin", "rb")
    retrieved_cred = loginFRetrieve.read()
    loginFRetrieve.close()
except OSError:
    print(RED + "Error retrieving credentials.")

# Decrypt the retrieved Postgresql creds and split into list
cipher = Fernet(retrieve_key())
credential = cipher.decrypt(retrieved_cred)
credential = credential.decode('utf-8')
split_creds = credential.split(":")


# Try connecting to Postgresql DB with decrypted credentials
def establish_connection(db_name:str):
    '''
    Tries to establish a connection to the specified database.
    Returns the connection object if successful.
    '''
    try:
        conn = psycopg2.connect(
            host=split_creds[0],
            dbname=db_name,
            user=split_creds[2],
            password=split_creds[3]
        )
    except psycopg2.OperationalError as error:
        print(RED + "Error:", error)
        print(YELLOW + "Exception TYPE:", type(error))
        return None
    else:
        return conn

"""Module to handle the logging of events on the system."""

from datetime import datetime # python lib to query date and time
import psycopg2
import dbconnection as dbc

def auth_log(log_type:str, uid:int) -> bool:
    '''
    Function to create an authentication event log entry in the database.
    Takes as input the user_id that actioned the event, as well as the operation type
    of the log.
    Type can be: Successful Login, Password Change, Account Locked, Locked Account
    Login Attempt.
    Returns True if log was created successfully and false if not.
    '''
    conn = dbc.establish_connection('eventlog')
    cursor = conn.cursor() # Connect Cursor to Eventlog DB
    dt_now = datetime.now()
    datestamp = dt_now.strftime("%d/%m/%Y %H:%M:%S") # captures datetime when the
    function is called
```

```
442         sql = """
443               INSERT INTO authlogs(datetime, operation, user_id)
444               VALUES (%(datetime)s,%(operation)s,%(uid)s)
445               """
446         val = {'datetime':datestamp,'operation':log_type, 'uid':uid}
447         try:
448             cursor.execute(sql,val)
449             conn.commit()
450         except psycopg2.OperationalError:
451             return False
452         return True
453 
454 
455     def operation_log( #pylint: disable=too-many-arguments
456         log_type:str, uid:int, source_id:int, modified=None, old_val=None, new_val=None
457     ) -> bool:
458         '''
459         Function to create an operations event log entry in the database.
460         Input: user_id that actioned the event, the operation type of the log, the id of
            the source,
461         the modified attribute and the before and after value of the attribute.
462         The operation type can be: View Source, Edit Source and Create Source.
463         Returns True if log was created successfully and false if not.
464         '''
465         conn = dbc.establish_connection('eventlog')
466         cursor = conn.cursor() # Connect Cursor to Eventlog DB
467         dt_now = datetime.now()
468         datestamp = dt_now.strftime("%d/%m/%Y %H:%M:%S") # captures datetime when function
            is called
469 
470         sql = """
471               INSERT INTO operationlogs
472               (datetime, operation, user_id, source_id, modified_attribute, old_value,
                  new_value)
473               VALUES(%(datetime)s,%(operation)s,%(uid)s,%(sid)s,%(attr)s,%(old)s,%(new)s)
474               """
475         val = {
476             'datetime':datestamp,
477             'operation':log_type,
478             'uid':uid, 'sid':source_id,
479             'attr':modified,'old':old_val,
480             'new':new_val
481             }
482         try:
483             cursor.execute(sql,val)
484             conn.commit()
485         except psycopg2.OperationalError:
486             return False
487         return True
488 
489 
490     def admin_log( #pylint: disable=too-many-arguments
491         log_type:str, admin_id:int, user_id:int, modified=None, old_val=None, new_val=None
492     ) -> bool:
493         '''
494         Function to create an admin event log entry in the database.
495         Input: admin's id that actioned the event, the type of operation, the effected user
            id,
496         the modified attribute and the before and after value of the attribute.
497         The operation type can be: Create User, Unlock User, Deactivate User and Edit User.
498         Returns True if log was created successfully and false if not.
499         '''
500         conn = dbc.establish_connection('eventlog')
501         cursor = conn.cursor() # Connect Cursor to Eventlog DB
502         dt_now = datetime.now()
503         datestamp = dt_now.strftime("%d/%m/%Y %H:%M:%S") # captures datetime when function
            is called
```

```
        sql = """
              INSERT INTO adminlogs
              (datetime, operation, admin_id, user_id, modified_attribute, old_value,
              new_value)
              VALUES(%(datetime)s,%(operation)s,%(adid)s,%(uid)s,%(attr)s,%(old)s,%(new)s)
              """
        val = {
            'datetime':datestamp,
            'operation':log_type,
            'adid':admin_id,
            'uid':user_id,
            'attr':modified,
            'old':old_val,
            'new':new_val
            }
        try:
            cursor.execute(sql,val)
            conn.commit()
        except psycopg2.OperationalError:
            return False
        return True

    """Module to define the Interface Class including menu options and user inputs."""

    import sys
    import stdiomask
    from validator_collection import checkers
    import authentication as auth
    import admin_operations as adops
    import operations as ops
    import eventlog as log


    RED = '\033[91m' # Error Messages
    GREEN = '\033[92m' # Success Messages
    BLUE = '\033[94m' # MOTD and Menus
    WHITE = '\033[97m' # User Input
    YELLOW = '\033[93m' # Notices to User
    BOLD = '\033[1m'


    class Interface: #pylint: disable=too-many-public-methods
        """Class that provides user menus and inputs. Differentiates views depending on
        user role."""

        def __init__(self):
            # Initialise the interface object when main.py is run.
            self.uid = None
            self.urole = None
            self.first_name = None
            self.username = None

            self.entered_username = None
            self.failed_attempts = 0

            self.motd() # call motd to display


        def motd(self):
            '''
            Display the motd including privacy and data policies. Prompts user to accept
            the ToS.
            If user agrees, display login prompt. If user disagrees, terminate CLI.
            '''
            with open('config/banner.bin','r') as file:
                motd = file.readlines()
                for line in motd:
```

```
568                 print(BLUE + BOLD + line, end='')
569         print(YELLOW+'Terms of Service: '+WHITE+
570 
                'https://marziohr.github.io/SSD_Project/policies/Terms%20and%20Conditions.pdf')
571         print(YELLOW+'Privacy Policy: '+WHITE+
572               'https://marziohr.github.io/SSD_Project/policies/Privacy%20Policy.pdf\n')
573         choice=self.y_n_input("Do you agree with the Terms of Service and Privacy Policy? (y/n): ")
574         if choice == 'y':
575             self.login()
576         else:
577             sys.exit()
578 
579 
580     def login(self):
581         '''
582         Asks user to enter username and password. If combination is found,
583         logs user in and saves the user id, role and first name.
584         If combination is incorrect after third try, user will be locked from logging in again.
585         '''
586         inpt_username = self.username_input()
587         inpt_password = stdiomask.getpass()
588 
589         login = auth.existing_user(inpt_username,inpt_password)
590 
591         if login is None: # Condition if Username was not found
592             print(RED + 'The Username and Password combination you have entered is incorrect.')
593             self.login()
594 
595         elif login is False: # Condition if entered password was incorrect
596             if self.entered_username != inpt_username: # if username differs, reset attempts to 1
597                 self.entered_username = inpt_username
598                 self.failed_attempts = 1
599                 print(RED + 'The Username and Password combination you have entered is incorrect.')
600                 self.login()
601             else: # trigger lock if login for same user is failed 3 times in succession
602                 if self.failed_attempts < 2:
603                     self.failed_attempts += 1
604                     print(RED, end='')
605                     print('The Username and Password combination you have entered is incorrect.')
606                     self.login()
607                 else:
608                     adops.lock_user(adops.fetch_user_info(username=inpt_username)[0])
609                     print(RED+'Your account has been locked because it reached ', end='')
610                     print('a maximum amount of failed login attempts.')
611                     print('Please contact the system administrator team for further assistance.\n')
612                     print(WHITE)
613                     sys.exit()
614 
615         else:
616             log.auth_log("Successful Login", login[0]) # log successful login
617             self.uid = login[0]
618             self.urole = login[2]
619             self.first_name = login[1]
620             self.username = inpt_username
621 
622             last_login = auth.fetch_last_login(self.uid)
623             if last_login is None:
624                 print(GREEN+f'\nAccess Granted! Welcome to the System, {self.first_name}.')
```

```
                print(YELLOW+'\nDue to you logging into the system for the first time,
                ', end='')
                print('please change your own password.')
                while True: # while password is not changed iterate over password
                change prompt
                    changed_pswd = self.change_password()
                    if changed_pswd:
                        auth.update_last_login(self.uid) # updates last_login date stamp
                        print(YELLOW+'You will now be logged out. ', end='')
                        print('Please login with your new password to use the system.')
                        self.logout()

            else:
                print(GREEN + f'\nAccess Granted! Welcome back, {self.first_name}.')
                auth.update_last_login(self.uid) # updates last_login date stamp
                self.handle_main()


    def handle_main(self):
        '''
        Handler to display main menu options based on user's role.
        1 = Administrator
        2 = Specialist
        3 = Third-Party Authority
        '''
        if self.urole == 1:
            self.admin_menu()
        elif self.urole == 2:
            self.specialist_menu()
        elif self.urole == 3:
            self.authority_menu()
        else:
            print(RED + f"Error: User Role not set correctly. Current value set to:
            {self.urole}")


    def admin_menu(self):
        '''
        Displays main menu options for the administrator role.
        Depending on choice, will trigger the operation from the admin_operations module.
        Choices include: Creating a new user, modifying an existing user,
        deactivating (soft deleting) an existing user, unlocking a user, logout.
        '''
        print(BLUE + '\nPlease select what you want to do:')
        print(' 1. Create New User')
        print(' 2. Modify Existing User')
        print(' 3. Deactivate User')
        print(' 4. Unlock User')
        print(' 5. Logout' + WHITE)
        choice = self.choice_input(5)
        if choice == 1:
            self.create_user()
        elif choice == 2:
            self.modify_user()
        elif choice == 3:
            self.deactivate_user()
        elif choice == 4:
            self.unlock_user()
        else:
            self.logout()


    def specialist_menu(self):
        '''
        Displays main menu options for the specialist (employee) role.
        Depending on choice, will trigger the operation from the operations module.
        Choices include: Search existing sources, create a new source, logout.
```

```
689             '''
690             print(BLUE + '\nPlease select what you want to do:')
691             print(' 1. Search Source')
692             print(' 2. Create New Source Entry')
693             print(' 3. Change Password')
694             print(' 4. Logout' + WHITE)
695             choice = self.choice_input(4)
696             if choice == 1:
697                 self.search_sources()
698             elif choice == 2:
699                 self.create_source()
700             elif choice == 3:
701                 changed_pswd = self.change_password()
702                 if changed_pswd:
703                     print(YELLOW+'\nYou will now be logged out. ', end='')
704                     print('Please login with your new password to use the system.')
705                     self.logout()
706                 else:
707                     self.handle_main()
708             else:
709                 self.logout()
710 
711 
712         def authority_menu(self):
713             '''
714             Displays main menu options for the authority (third-party) role.
715             Depending on choice, will trigger the operation from the operations module.
716             Choices include: Search existing sources, logout.
717             '''
718             print(BLUE + '\nPlease select what you want to do:')
719             print(' 1. Search Source')
720             print(' 2. Change Password')
721             print(' 3. Logout' + WHITE)
722             choice = self.choice_input(3)
723             if choice == 1:
724                 self.search_sources()
725             elif choice == 2:
726                 changed_pswd = self.change_password()
727                 if changed_pswd:
728                     print(YELLOW+'\nYou will now be logged out. ', end='')
729                     print('Please login with your new password to use the system.')
730                     self.logout()
731                 else:
732                     self.handle_main()
733             else:
734                 self.logout()
735 
736 
737         def create_user(self):
738             '''
739             Prompts information and inputs for new user creation. If entered details pass
                validation,
740             calls admin_operations module to execute the creation on the database level.
741 
742             Validation rules:
743             First Name: >2 characters and may only contain letters, spaces and '-'
744             Last Name: >2 characters and may only contain letters, spaces and '-'
745             Date of Birth: Exactly 10 characters and may only contain numbers and '-'
746             Email: Must contain exactly 1x '@' and atleast 1x '.' and end with a letter.
747             May contain alphanum and '-', '.', '_', '+'
748 
749             If user is registered successfully, the user password will be autogenerated
750             and sent to the user's email address.
751             '''
752             print(BLUE + '\nCreate a New User')
753             print('-------------------------' + WHITE)
754             inpt_first = self.name_input("First")
```

```
755             inpt_last = self.name_input("Last")
756             inpt_email = self.email_input(register=True)
757             inpt_dob = self.dob_input()
758             print(BLUE + '\nPlease select the user role:')
759             print(' 1. Administrator')
760             print(' 2. Specialist')
761             print(' 3. External Authority' + WHITE)
762             inpt_role = self.choice_input(3)
763 
764 
                result=adops.register_new_user(inpt_first,inpt_last,inpt_dob,inpt_email,inpt_role
                ,self.uid)
765             if result:
766                 print(GREEN + "User has successfully been created!")
767             else:
768                 print(RED + "Error: User has not been created successfully. Please try
                    again.")
769 
770             choice = self.y_n_input(WHITE + "\nDo you want to create another user? (y/n): ")
771             if choice == 'y':
772                 self.create_user()
773             else:
774                 self.admin_menu()
775 
776 
777         def modify_user(self): #pylint: disable=too-many-branches
778             '''
779             Function to prompt user modification options.
780             Admin can change user's first name, last name, dob and user role.
781             '''
782             print(BLUE + '\nModify an existing User')
783             print('---------------------------' + WHITE)
784             inpt_email = self.email_input()
785             result = adops.fetch_user_info(email=inpt_email)
786             if result is not None:
787                 print(BLUE + '\nUser Found:')
788                 print(f'ID: {result[0]}\t|\tFirst Name: {result[1]}\t|\t', end='')
789                 print(f'Last Name: {result[2]}\t|\tEmail: {result[3]}\t|\tDate of Birth: ',
                    end='')
790                 print(f'{result[4]}\t|\tCurrent Status: {result[5]}\n')
791                 print('What would you like to change?')
792                 print(' 1.) First Name')
793                 print(' 2.) Last Name')
794                 print(' 3.) Date of Birth')
795                 print(' 4.) Cancel' + WHITE)
796                 edit_option = self.choice_input(4)
797                 if edit_option==1:
798                     inpt_first = self.name_input("new First")
799                     changed = adops.modify_user(result[0], 'first_name', inpt_first,
                        self.uid)
800                     if changed:
801                         print(GREEN + "User has successfully been modified!")
802                     else:
803                         print(RED + "Error: User has not been modified successfully. Please
                            try again.")
804                 elif edit_option==2:
805                     inpt_last = self.name_input("new Last")
806                     changed = adops.modify_user(result[0], 'last_name', inpt_last, self.uid)
807                     if changed:
808                         print(GREEN + "User has successfully been modified!")
809                     else:
810                         print(RED + "Error: User has not been modified successfully. Please
                            try again.")
811                 elif edit_option==3:
812                     inpt_dob = self.dob_input()
813                     changed = adops.modify_user(result[0], 'dob', inpt_dob, self.uid)
814                     if changed:
```

```
                    print(GREEN + "User has successfully been modified!")
                else:
                    print(RED + "Error: User has not been modified successfully. Please try again.")
        else:
            print(RED + 'No User found for the email address.')
        choice = self.y_n_input(WHITE + "\nDo you want to modify another user? (y/n): ")
        if choice == 'y':
            self.modify_user()
        else:
            self.handle_main()


    def deactivate_user(self):
        '''
        Prompts dialogue for deactivating an existing user.
        If successful, the user in question will have his status changed to 'deactivated'.
        '''
        print(BLUE + '\nDeactivate an Existing User')
        print('---------------------------' + WHITE)
        deact_email = self.email_input()
        result = adops.fetch_user_info(email=deact_email)
        if result is not None:
            if result[5] != 2:
                print(BLUE + 'User Found:')
                print(f'ID: {result[0]}\t|\tFirst Name: {result[1]}\t|\tLast Name: ', end='')
                print(f'{result[2]}\t|\tEmail: {result[3]}\t|\tCurrent Status: {result[5]}\n'+WHITE)
                choice = self.y_n_input("Are you sure you want to deactivate this user? (y/n): ")
                if choice == 'y':
                    deactivated = adops.deactivate_user(result[0], self.uid, result[5])
                    if deactivated:
                        print(GREEN + "User has successfully been deactivated.")
                    else:
                        print(RED + 'Error: User could not be deactivated. Please try again.')
            else:
                print(RED + 'Error: User is already deactivated.')
        else:
            print(RED + 'No User found for the email address.')
        choice = self.y_n_input(WHITE + "\nDo you want to deactivate another user? (y/n): ")
        if choice == 'y':
            self.deactivate_user()
        else:
            self.admin_menu()


    def unlock_user(self):
        '''
        Prompts dialogue for unlocking a locked-out user.
        If successful, the user in question will have his status changed back to 'active'
        '''
        print(BLUE + '\nUnlock an Existing User')
        print('---------------------------'+ WHITE)
        unlock_email = self.email_input()
        result = adops.fetch_user_info(email=unlock_email)
        if result is not None:
            if result[5] == 3:
                print(BLUE + 'User Found:')
                print(f'ID: {result[0]}\t|\tFirst Name: {result[1]}\t|\tLast Name: ', end='')
                print(f'{result[2]}\t|\tEmail: {result[3]}\t|\tCurrent Status: {result[5]}\n'+WHITE)
```

```
873                 choice = self.y_n_input("Are you sure you want to unlock this user?
                    (y/n): ")
874                 if choice == 'y':
875                     unlocked = adops.unlock_user(result[0], self.uid)
876                     if unlocked:
877                         print(GREEN, end='')
878                         print("User has successfully been unlocked! The User can now
                            login again.")
879                     else:
880                         print(RED + 'Error: User could not be unlocked. Please try
                            again.')
881             else:
882                 print(RED + 'Error: User is currently not locked.')
883         else:
884             print(RED + 'No User found for the email address.')
885         choice = self.y_n_input(WHITE + "\nDo you want to unlock another user? (y/n): ")
886         if choice == 'y':
887             self.unlock_user()
888         else:
889             self.admin_menu()
890
891
892     def search_sources(self): #pylint: disable=too-many-branches,too-many-statements
893         '''
894         Displays options for Source Search. Once a Source has been selected, will
            display the
895         main information regarding the source, as well as provide the option to modify
            it.
896         '''
897         print(BLUE + '\nSearch Source')
898         print('--------------------------')
899         print('\nPlease select the field you want to search:')
900         print(' 1. Name')
901         print(' 2. Url')
902         print(' 3. Description')
903         print(' 4. Threat Level' + WHITE)
904         input_field = self.choice_input(4)
905         if input_field == 4:
906             search_term = self.choice_input(5)
907         else:
908             search_term = self.search_string_input("Please enter text to search")
909         field_name = self.map_input_field(input_field)
910         result = ops.search_for_source(field_name, search_term)
911
912         if len(result) == 0:
913             print(RED + "No sources found")
914             self.search_sources()
915
916         print(BLUE + "\nId\tName")
917         print("--\t--------------")
918         for item in result:
919             print((str(item[0]) + "\t" +item[1]))
920
921         print(WHITE + '\nPlease enter the Source Id to view details:')
922         selected_id = self.source_id_input()
923         # Check if entered Source Id is valid
924         is_valid_id = False
925         for item in result:
926             if item[0] == selected_id:
927                 log.operation_log("View Source", self.uid, selected_id) # log view
                    source event
928                 is_valid_id = True
929         if not is_valid_id:
930             print(RED + "Invalid source Id")
931             self.search_sources()
932         source_details = ops.get_source_by_id(selected_id)
933
```

```
            if source_details is None:
                print (RED + "Error occured")
                self.search_sources()

            print (BLUE + "\nId : " + str(source_details[0]))
            print ("Name : " + source_details[1])
            print ("Url : " + source_details[2])
            print ("\nDescription : \n" + source_details[4])
            print ("\nThreat Level : " + str(source_details[3]))
            print ("Created Date : " + source_details[5].strftime("%m/%b/%Y"))
            print ("Modified Date : " + source_details[6].strftime("%m/%b/%Y"))

            print(YELLOW + '\nPlease select what you want to do:')

            if self.urole == 3:
                print(' 1. Search new source')
                print(' 2. Main menu' + WHITE)
            else:
                print(' 1. Edit')
                print(' 2. Search new source')
                print(' 3. Main menu' + WHITE)

            choice = self.choice_input(3)

            if self.urole == 3:
                if choice == 1:
                    self.search_sources()
                elif choice == 2:
                    self.specialist_menu()
            else:
                if choice == 1:
                    print(BLUE + '\nPlease select the field you want to edit:')
                    print(' 1. Name')
                    print(' 2. Url')
                    print(' 3. Description')
                    print(' 4. Threat Level' + WHITE)
                    input_edit_field = self.choice_input(4)

                    if input_edit_field == 2:
                        new_value = self.source_create_url_input("Please enter new url")
                    elif input_edit_field == 4:
                        new_value = self.choice_input(5)
                    else:
                        new_value = self.search_string_input(WHITE + "Please enter new
                        value")

                    edit_field_name = self.map_input_field(input_edit_field)
                    ops.modify_source(int(selected_id), edit_field_name, new_value, self.uid)
                    print(GREEN + '\nSource has been modified successfully')
                    self.specialist_menu()

                elif choice == 2:
                    self.search_sources()
                elif choice == 3:
                    self.specialist_menu()


        def create_source(self):
            '''
            Displays options to enter a new source into the system.
            If validations are met, will action the creation using the operations module.
            '''
            print(BLUE + '\nCreate a New Source')
            print('--------------------------' + WHITE)
            inpt_name = self.source_create_string_input("Please enter the Source Name")
            inpt_url = self.source_create_url_input("Please enter the Source Url")
            inpt_description = self.source_create_string_input("Please enter the Source
```

```
            Description")
1000
1001        print(YELLOW + '\nPlease enter the threat level:')
1002        print(' 0 : Min - 5 : Max' + WHITE)
1003        inpt_threat_level = self.choice_input(5)
1004
1005
            result=ops.create_new_source(inpt_name,inpt_url,inpt_threat_level,inpt_descriptio
            n,self.uid)
1006        if result:
1007            print(GREEN + "Source has successfully been created!")
1008        else:
1009            print(RED + "Error: Source has not been created successfully. Please try
                again.")
1010
1011        choice = self.y_n_input(WHITE + "\nDo you want to create another Source? (y/n):
            ")
1012        if choice == 'y':
1013            self.create_source()
1014        else:
1015            self.specialist_menu()
1016
1017
1018    def change_password(self) -> bool:
1019        '''
1020        Function and prompt to change password. User will need to enter his existing
            password.
1021        User will then have to enter the new password and confirm it.
1022        If all correct, password will be updated on the database.
1023        Returns True if successful and False if not.
1024        '''
1025        inpt_password = stdiomask.getpass(WHITE+'Please enter your current Password: ')
1026        login = auth.existing_user(self.username,inpt_password)
1027
1028        if login is None or login is False: # Condition if Password was not found or is
            incorrect
1029            print(RED + 'The Password you have entered is incorrect. Please check and
                try again.')
1030            return False
1031
1032        print(YELLOW+'\nPlease Note: Your password must be at least 12 characters
            ',end='')
1033        print('long, include letters and numbers, as well as atleast one special
            character.')
1034        new_password = stdiomask.getpass(WHITE + '\nPlease enter your new Password: ')
1035        confirm_password = stdiomask.getpass('Please confirm your new Password: ')
1036        valid_pswd = self.password_validator(new_password)
1037
1038        if valid_pswd:
1039            if new_password == inpt_password:
1040                print(RED+'\nError: Your new password may not be the same as your old
                    one.')
1041                return False
1042            if new_password == confirm_password:
1043                changed = ops.change_password(self.uid, auth.hash_pswd(new_password))
1044                if changed:
1045                    print(GREEN + '\nYour password has been successfully updated.')
1046                    return True
1047                print(RED + '\nError: Your password could not be updated.', end='')
1048                print('Please check with the Administrator Team for further
                    instructions.')
1049                return False
1050            print(RED + '\nError: Your entered passwords do not match. Please try
                again.')
1051            return False
1052        print(RED)
1053        print("Error: Your password does not confirm with the system's password
```

```
            standards.")
            return False


    def logout(self):
        '''Message to be displayed when logout is chosen.'''
        print(BLUE, end='')
        print("\nThank you for using the NCSC Suspect Sources System. ", end='')
        print(f"See you soon, {self.first_name}!\n" + WHITE)
        sys.exit()


    def choice_input(self, num_choices:int) -> int:
        '''
        Wrapper to validate the user input for a menu selection.
        The amount of different options to choose from can be set with the argument
        "num_choices".
        Returns the chosen option as an Integer.
        '''
        user_input = input(WHITE + "\nSelect option: ")
        try:
            int_input = int(user_input)
        except ValueError:
            print(RED + "Error: Invalid selection. Please check your input and try
            again.")
            return self.choice_input(num_choices)
        if 0 < int_input <= num_choices:
            return int_input
        print(RED + "Error: Invalid selection. Please check your input and try again.")
        return self.choice_input(num_choices)


    def username_input(self) -> str:
        '''
        Wrapper to validate and sanitise the user input for username.
        Ensures entered string is following the validation rules. If so, returns the
        entered string.
        '''
        valid_char = ('.','_','-')
        min_len = 5

        input_user = input(WHITE + "\nPlease enter your Username: ")

        if len(input_user) < min_len: # checks the length of the entered username
            print(RED + "Error: Entered username is invalid. Please check and try
            again.")
            return self.username_input()
        for char in input_user: # checks each character of user input
            if char.isalnum():
                continue # continue if current char is either alpha or numerical
            if char in valid_char:
                continue # continue if current char is part of the valid characters tuple
            print(RED + "Error: Entered username is invalid. Please check and try
            again.")
            return self.username_input()
        return input_user # returns entered string if all validation rules are met


    def name_input(self, name_type:str) -> str:
        '''
        Wrapper to validate and sanitise the user input for first and lastname.
        Ensures entered string is following the validation rules. If so, returns the
        entered string.
        Validation: >2 characters and may only contain letters, spaces and '-'
        The argument "name_type" defines whether the entered name is a first or last
        name.
        "first" = first name
```

```
1113            "last" = last name
1114            '''
1115            valid_char = (' ','-')
1116            min_len = 3
1117
1118            input_name = input(WHITE + f"\nPlease enter the User's {name_type} Name: ")
1119
1120            if len(input_name) < min_len: # checks the length of the entered name
1121                print(RED + "Error: Entered Name is invalid. Please check and try again.")
1122                return self.name_input(name_type)
1123            for char in input_name: # checks each character of name input
1124                if char.isalpha():
1125                    continue # continue if current char is a letter
1126                if char in valid_char:
1127                    continue # continue if current char is part of the valid characters tuple
1128                print(RED + "Error: Entered Name is invalid. Please check and try again.")
1129                return self.name_input(name_type)
1130            return input_name # returns entered string if all validation rules are met
1131
1132
1133        def email_input(self, register=False) -> str:
1134            '''
1135            Wrapper to validate and sanitise the user input for email.
1136            Ensures entered string is following the validation rules. If so, returns the
                entered string.
1137            Validation: Must contain exactly 1x '@', atleast 1x '.' and end with a letter.
1138            May contain alnum and '-', '.', '_', '+'
1139            '''
1140            valid_char = ('-', '.', '_', '+')
1141            min_len = 7
1142            num_at_sign = 0
1143            contains_dot = False
1144            ends_with_letter = False
1145
1146            input_email = input(WHITE + "\nPlease enter the User's Email Address: ")
1147
1148            if register:
1149                email_exists = adops.fetch_user_info(email=input_email)
1150                if email_exists:
1151                    print(RED + "Error: Entered Email Address is already tied to a User in
                        the system.")
1152                    return self.email_input()
1153
1154            if len(input_email) < min_len:
1155                print(RED + "Error: Entered Email Address is invalid. Please check and try
                    again.")
1156                return self.email_input()
1157
1158            if input_email[-1].isalpha(): # checks if email ends with a letter
1159                ends_with_letter = True
1160
1161            if '.' in input_email: # checks if email contains atleast one '.'
1162                contains_dot = True
1163
1164            for char in input_email: # checks each character of name input
1165                if char.isalnum():
1166                    continue # continue if character is a letter or number
1167                if char in valid_char:
1168                    continue # continue if character is part of the valid_char tuple
1169                if char == '@':
1170                    num_at_sign += 1 # counts the number of times the '@' sign appears
1171                    continue
1172                print(RED + "Error: Entered Email Address is invalid. Please check and try
                    again.")
1173                return self.email_input()
1174
1175            if num_at_sign == 1 and contains_dot and ends_with_letter:
```

```
1176                return input_email # if all validations are met, the input string is returned
1177            print(RED + "Error: Entered Email Address is invalid. Please check and try
                again.")
1178            return self.email_input()
1179
1180
1181        def dob_input(self) -> str:
1182            '''
1183            Wrapper to validate and sanitise the user input date of birth.
1184            Ensures that string is following validation rules. If so, returns the entered
                string.
1185            Validation: Exactly 10 characters and may only contain numbers and '-'
1186            '''
1187            valid_char = ('-')
1188            exact_len = 10
1189
1190            input_dob = input(WHITE + "\nPlease enter the User's Date of Birth (Format
                YYYY-MM-DD): ")
1191
1192            if len(input_dob) != exact_len:
1193                print(RED + "Error: Entered Date of Birth is invalid. Please check and try
                    again.")
1194                return self.dob_input()
1195            for char in input_dob:
1196                if char.isnumeric():
1197                    continue
1198                if char in valid_char:
1199                    continue
1200                print(RED + "Error: Entered Date of Birth is invalid. Please check and try
                    again.")
1201                return self.dob_input()
1202            return input_dob
1203
1204
1205        def password_validator(self, password:str) -> bool: #pylint: disable=no-self-use
1206            '''
1207            Checks if a given password is conform to the system's standards.
1208            A password must be atleast 12 characters long, include letters and numbers,
1209            and atleast one special character. Returns True if conform and False if not.
1210            '''
1211            min_len = 12
1212            includes_letter = False
1213            includes_number = False
1214            includes_special = False
1215
1216            valid_special_char = '[@_!#$%^&*()<>?/\}{~:;]-.,' #pylint:
                disable=anomalous-backslash-in-string
1217
1218            if len(password) < min_len:
1219                return False
1220            for char in password:
1221                if char.isalpha():
1222                    includes_letter = True
1223                elif char.isnumeric():
1224                    includes_number = True
1225                elif char in valid_special_char:
1226                    includes_special = True
1227            if includes_special and includes_number and includes_letter:
1228                return True
1229            return False
1230
1231
1232        def y_n_input(self, question:str) -> str:
1233            '''Validates a yes/no question and returns the str if answer is either 'y' or
                'n'.'''
1234            input_choice = input(WHITE + question).lower()
1235            if input_choice in ('y', 'n'):
```

```
            return input_choice
        print(RED+"Error: Please answer either 'y' for 'yes' or 'n' for 'no'.")
        return self.y_n_input(question)


    def source_create_url_input(self, messege) -> str:
        '''Validates url input for source creation.'''
        input_text = input(WHITE+f"\n{messege}: ")
        if checkers.is_url(input_text) is False:
            print(RED + "Error: Entered data is invalid. Please check and try again.")
            return self.source_create_url_input(messege)
        return input_text # returns entered string if all validation rules are met


    def source_create_string_input(self, messege) -> str:
        '''Validates string user input for source creation.'''
        min_len = 5
        input_text = input(WHITE+f"\n{messege}: ")
        if len(input_text) < min_len: # checks the length of the entered text
            print(RED + "Error: Entered data is invalid. Please check and try again.")
            return self.source_create_string_input(messege)
        return input_text # returns entered string if all validation rules are met


    def search_string_input(self, messege) -> str:
        '''Validates string user input search term.'''
        min_len = 3
        input_text = input(WHITE + f"\n{messege}: ")
        if len(input_text) < min_len: # checks the length of the entered text
            print(RED + "Error: Search term should be more than or equal to three
            characters.")
            return self.search_string_input(messege)
        return input_text # returns entered string if all validation rules are met


    def source_id_input(self) -> int:
        '''Validates User Input for Source Id.'''
        user_input = input(WHITE + "\nSelect Id: ")
        try:
            int_input = int(user_input)
        except ValueError:
            print(RED + "Error: Invalid id. Please check your input and try again.")
            return self.source_id_input()
        return int_input


    def map_input_field(self, input_field:int) -> str: #pylint: disable=no-self-use
        '''Map user input integer value to database field.'''
        if input_field == 1:
            return "name"
        if input_field == 2:
            return "url"
        if input_field == 3:
            return "description"
        return "threat_level"
```

```
"""Main module to execute the Suspect Sources CLI."""

import interface

if __name__ == "__main__":
    main_cli = interface.Interface()
```

```
"""Module to handle notification actions for New Users, Password Changes and Source
Additions."""

import smtplib
```

```
from email.mime.text import MIMEText
from cryptography.fernet import Fernet
import dbconnection as dbc

RED = '\033[91m' # Erorr Messages
GREEN = '\033[92m' # Success Messages

OUTBND_EMAIL =  'suspect.sources@gmail.com'
OUTBND_ENC_PSWD = b'gAAAAABgbavwJiy3quTfBs44koynkhs5sNYVETrSeh-aTlFl3HH8LSMvtC0-09fkvqdyTgJJ6DCbmD3nr4R6V5E7VSmtbwh8GVqTqVRU1S4LoJjM0rSPuyo='

with open('config/email_body.html','r') as file:
    HTML_BODY = file.readline()

retrieved_key = dbc.retrieve_key()
cipher = Fernet(retrieved_key)
OUTBND_PSWD = cipher.decrypt(OUTBND_ENC_PSWD)
OUTBND_PSWD = OUTBND_PSWD.decode('utf-8')

def registration_email(firstname:str, email:str, username:str, password:str) -> bool:
    '''
    Function to trigger the registration email to a new system user.
    Takes as input the firstname, email and autogenerated password.
    Sends a notification to the given email containing the password for the created user.
    '''
    subject = f'Welcome to the NCSC Suspect Sources System, {firstname}!'
    email_body = f"""
                <p><strong>Welcome, {firstname}</strong>!</p>
                <p>You may now log into the Suspect Sources Interface using the following credentials:</p>
                <p><span style="color: #000080;"><strong>Username:</strong></span> <strong>{username}</strong><br />
                <span style="color: #000080;"><strong>Password:</strong></span> <strong>{password}</strong></p>
                <p>Please note, that you will be prompted to change your password once you log into the system for the first time.
                This is done to enhance the security of your account.<br />
                Please don't hesitate to contact a system administrator or the technical support team should you run into any difficulties.</p>
                """
    my_email = MIMEText(HTML_BODY.replace('{CONTENT}', email_body), "html")
    my_email["From"] = OUTBND_EMAIL
    my_email["To"] = email
    my_email["Subject"] = subject
    try:
        server = smtplib.SMTP("smtp.gmail.com")
        server.starttls()
        server.login(user=OUTBND_EMAIL, password=OUTBND_PSWD)
        server.sendmail(OUTBND_EMAIL, email, my_email.as_string())
    except smtplib.SMTPException:
        return False
    return True


def new_source_email(
    recipient_list:list, source_id:id, source_name:str, source_url:str, source_threat_level:int
) -> bool:
    '''
    Function to trigger the notification when a new suspect source has been added to the system.
    Takes as input a list of tuples containing the email:firstname pair
    of all users in the system with the role "Authority".
    Also takes the information of the added source, such as the id, name, url and threat level.
    Sends a notification to all emails in the tuple with details of the newly added
```

```
        suspect source.
        Please note recipient_list structure should be:
        [('email1@email.com', 'firstname1'), ('email2@email.com', 'firstname2')]
        '''
        subject = f'New Suspect Source has been added: {source_name}'
        email_body = f"""
                    <p><strong>Dear FIRSTNAME,</strong></p>
                    <p>This email serves as a notification that a new suspect source has
                    been added to the NCSC Suspect Sources System.</p>
                    <p>Overview of the added source:</p>
                    <p><span style="color: #0000ff;"><strong>Source Name:</strong></span>
                    <strong>{source_name}</strong><br />
                    <span style="color: #0000ff;"><strong>Source ID:</strong></span>
                    <strong>{source_id}</strong><br /> <strong>
                    <span style="color: #0000ff;"><strong>Source URL:</strong></span>
                    {source_url}<br /> <strong>
                    <span style="color: #0000ff;">Source Threat Level:</span>
                    {source_threat_level}</strong></p>
                    <p>Please log into the system and search for the threat id to obtain a
                    full description of the newly added suspect source.</p>
                    <p>Please don't hesitate to contact a system administrator or the
                    technical support team should you run into any difficulties.</p>
                    """
        for item in recipient_list:
            email = item[0]
            first_name = item[1]
            content = email_body.replace('FIRSTNAME', first_name)
            my_email = MIMEText(HTML_BODY.replace('{CONTENT}', content), "html")
            my_email["From"] = OUTBND_EMAIL
            my_email["To"] = email
            my_email["Subject"] = subject
            try:
                server = smtplib.SMTP("smtp.gmail.com")
                server.starttls()
                server.login(user=OUTBND_EMAIL, password=OUTBND_PSWD)
                server.sendmail(OUTBND_EMAIL, email, my_email.as_string())
            except smtplib.SMTPException:
                return False
            print(GREEN + f"Sent Notification successfully to: {first_name} at {email}") #
            Debug Line to check email sends
        return True


    def changed_password_email(firstname:str, email:str) -> bool:
        '''
        Function to trigger a notification email if the password of a user is changed.
        Takes as input the firstname and email of the user.
        Will send an email to the email given as an arg to confirm that the password has
        been changed.
        '''
        subject = 'Password Changed Successfully'
        email_body = f"""
                    <p><strong>Dear {firstname},</strong></p>
                    <p>This email is to notify you of a successful password change to your
                    NCSC Suspect Sources account.
                     You are now able to log in with your username and newly set
                     password.</p>
                    <p>If you did not action this change, please contact the NCSC
                    Administrator Team immediately. In this case, your account might be
                    comprised.</p>
                    """
        my_email = MIMEText(HTML_BODY.replace('{CONTENT}', email_body), "html")
        my_email["From"] = OUTBND_EMAIL
        my_email["To"] = email
        my_email["Subject"] = subject
        try:
            server = smtplib.SMTP("smtp.gmail.com")
```

```
            server.starttls()
            server.login(user=OUTBND_EMAIL, password=OUTBND_PSWD)
            server.sendmail(OUTBND_EMAIL, email, my_email.as_string())
        except smtplib.SMTPException:
            return False
        return True

    """Module to execute user operations for Specialist and Authority Role."""

    from datetime import datetime
    import psycopg2
    from psycopg2 import sql
    import dbconnection as dbc
    import eventlog as log
    import notification
    import admin_operations as adops


    def search_for_source(attribute:str, value:str) -> list:
        '''
        Function to query a specific search on the sources table.
        Two arguments are being passed: attribute and value.
        The attribute indicates the column to search for (e.g. by ID or by Name).
        The value indicates the value to search for within the column.
        Example: search_for_source('name', 'Google') -> would search for 'Google' in the
        'name' column
        The return value contains all results from the query (i.e. the 'fetchall()' result)
        '''
        cursor = dbc.establish_connection('data').cursor()
        if attribute == "threat_level":
            value = int(value)
            stmt=sql.SQL(
            "SELECT id, name FROM sources WHERE {attribute} = {value} order by id"
            ).format(
                attribute = sql.Identifier(attribute.lower()),
                value = sql.Literal(value),
            )
        else:
            value = value.lower()
            value = '%'+value+'%'
            stmt=sql.SQL(
            "SELECT id, name FROM sources WHERE lower({attribute}) like {value} order by id"
            ).format(
                attribute = sql.Identifier(attribute.lower()),
                value = sql.Literal(value),
            )

        cursor.execute(stmt)
        result = cursor.fetchall()
        output = []
        for item in result:
            output.append((item[0], item[1]))
        return output

    def get_source_by_id(source_id:int) -> list:
        '''
        Function to return source information by its id.
        Used in the search operation of the interface module.
        Takes as argument the source id and returns a tuple:
        (id, name, url, threat level, description, creation date, modified date)
        '''
        cursor = dbc.establish_connection('data').cursor()
        psql = """
              SELECT id, name, url, threat_level, description, creation_date, modified_date
              FROM sources WHERE id = %(value)s
              """
        val = {'value': source_id}
```

```
    try:
        cursor.execute(psql, val)
        result = cursor.fetchall()[0]
    except IndexError:
        return None
    return (result[0], result[1], result[2], result[3], result[4], result[5], result[6])


def create_new_source(name:str, url:str, threat_level:int, description:str, uid:int) -> bool:
    '''
    Function to create a new entry in the sources database table.
    Takes as input the name of the source, the url, the threat level and the description.
    Returns bool True/False depending on whether the creation was successful.
    If successful, triggers email notification to all users with role=3 (External Authority).
    '''
    conn = dbc.establish_connection('data')
    cursor = conn.cursor()
    psql = """
          INSERT INTO sources (name, url, threat_level, description, creation_date, modified_date)
          VALUES
          (%(name)s,%(url)s,%(threat_level)s,%(description)s,%(creation_date)s,%(modified_date)s)
          RETURNING id;
          """
    val = {
        'name':name,
        'url':url,
        'threat_level':threat_level,
        'description':description,
        'creation_date':datetime.now(),
        'modified_date':datetime.now()
        }
    try:
        cursor.execute(psql, val)
        conn.commit()
    except psycopg2.OperationalError:
        return False
    else:
        recipients = adops.fetch_all_authorities() # fetching list of authority users
        source_id = cursor.fetchone()[0] # retrieving the id of the newly created source
        log.operation_log("Create Source", uid, source_id) # log source creation event
        notification.new_source_email(recipients, source_id, name, url, threat_level) # notification
        return True


def modify_source(source_id:int, attribute:str, new_value:str, uid:int) -> bool:
    '''
    Function to modify the information of an existing source.
    Takes as input the id of the source that is being modified, the attribute to be modified
    and the new value that should be saved.
    Example: modify_source(1, 'name', 'Google') -> Changes the 'name' of the source id=1 to 'Google'
    Returns bool True/False depending on whether the modification was successful.
    '''
    conn = dbc.establish_connection('data')
    cursor = conn.cursor()

    stmt = sql.SQL("SELECT {attribute} FROM sources WHERE id = {sid}").format(
            attribute = sql.Identifier(attribute),
            sid = sql.Literal(source_id),
    )
    cursor.execute(stmt)
```

```
1535        curr_val = cursor.fetchall()[0][0]
1536
1537        if attribute == 'threat_level':
1538            new_value = int(new_value) # change new value to int type if Threat Level is
                being changed
1539
1540        stmt = sql.SQL(
1541            "UPDATE sources SET {attribute}={value}, modified_date={dtnow} WHERE id = {sid}"
1542        ).format(
1543                attribute = sql.Identifier(attribute),
1544                sid = sql.Literal(source_id),
1545                value = sql.Literal(new_value),
1546                dtnow = sql.Literal(datetime.now()),
1547        )
1548        try:
1549            cursor.execute(stmt)
1550            conn.commit()
1551        except psycopg2.OperationalError:
1552            return False
1553        log.operation_log(
1554            "Edit Source",
1555            uid,
1556            source_id,
1557            modified=attribute,
1558            old_val=str(curr_val),
1559            new_val=str(new_value)
1560            ) # log edit source event
1561        return True
1562
1563
1564    def change_password(user_id:int, new_password:str) -> bool:
1565        '''
1566        Function to change the password of a specific user. Input: user id and new password
            hash.
1567        Returns a bool value depending on whether the modification was successful.
1568        If successful, triggers email notification to user that password has changed.
1569        '''
1570        conn = dbc.establish_connection('authentication')
1571        cursor = conn.cursor()
1572        psql = "UPDATE users SET password = %(val)s WHERE id = %(id)s;"
1573        val = {'val':new_password, 'id': user_id}
1574        try:
1575            cursor.execute(psql,val)
1576            conn.commit()
1577        except psycopg2.OperationalError:
1578            return False
1579        log.auth_log("Password Change", user_id)
1580        fetch_user = adops.fetch_user_info(uid=user_id) # retrieves user's information
1581        u_email = fetch_user[3]
1582        u_first_name = fetch_user[1]
1583        notification.changed_password_email(u_first_name, u_email) # triggers email
            notification
1584        return True
1585
1586
```