```
 1 """
 2 Request functions available to administrator users.
 3 """
 4 import security.password as phandler
 5
 6
 7 class AdminRequestHandler:
 8     def __init__(self, postgres_handler):
 9         self.postgres_handler = postgres_handler
10
11     def get_students(self, query=None):
12         """
13             Function called to get all students.
14
15             :returns JSON with format
16
17                 successful: boolean
18                 students: [student dictionary object]
19         """
20         sql = "SELECT * FROM student"
21         results = self.postgres_handler.select(sql)
22         results_formatted = []
23         for student in results:
24             results_formatted.append({'name': student[1], 'email':
   student[3], 'id': str(student[0])})
25         print(results_formatted)
26         return {
27             'students': results_formatted,
28             'successful': True
29         }
30
31     def get_student_location(self, student_id):
32         """
33             Function called to get all locations given a student. Accepts
    JSON with
34             format:
35
36                 student_id: String
37
38             :returns JSON with format
39
40                 successful: boolean
41                 results: [location dictionary object]
42         """
43         if student_id:
44             # Precondition: Ensure that student_id is not equal to None
45             # Create SQL query and prepare it to be concatenated
46             sql = "SELECT * FROM location WHERE student_student_id = (%s
   );"
```

```
47              student_id = (student_id,)
48              locations = self.postgres_handler.query(sql, student_id)
49              locations_formatted = []
50              if locations:
51                  for location in locations:
52                      sql = "SELECT * FROM beacon WHERE beacon_id = (%s);"
53                      beacon_id = (str(location[2]),)
54                      beacon = self.postgres_handler.query(sql, beacon_id)[
   0]
55
56                      sql = "SELECT * FROM zone WHERE zone_id = (%s);"
57                      zone_id = (str(beacon[5]),)
58                      zone = self.postgres_handler.query(sql, zone_id)[0]
59
60                      locations_formatted.append({'room_number': beacon[1],
    'zone_id': str(zone[0]), 'description': str(beacon[2]), 'timestamp':
   location[1]})
61              return {
62                  'results': locations_formatted,
63                  'successful': True
64              }
65          return {
66              'successful': False,
67              'reason': 'No student_id provided'
68          }
69
70      def get_zones(self, query):
71          """
72              Function called to get all zones.
73
74              :returns JSON with format
75
76                  successful: boolean
77                  zones: [zone dictionary object]
78          """
79          sql = "SELECT zone_name FROM zone"
80          results = self.postgres_handler.select(sql)
81          results_formatted = []
82          for zone in results:
83              results_formatted.append(str(zone[0]))
84
85          print(results_formatted)
86          return {
87              'zones': results_formatted,
88              'successful': True
89          }
90
91      def get_beacons(self, query):
92          """
```

```
 93             Function called by a IOS admin to get all beacons.
 94
 95             :returns JSON with format
 96
 97                 successful: boolean
 98                 beacons: [beacon dictionary object]
 99         """
100         sql = "SELECT * FROM beacon"
101         beacons = self.postgres_handler.select(sql)
102         results_formatted = []
103         for beacon in beacons:
104             # We need zone names as well, all we have are the zone_ids
105             sql = "SELECT zone_name FROM zone WHERE zone_id = %s"
106             zone_id = (beacon[5],)
107             zone_name = self.postgres_handler.query(sql, zone_id)[0][0]
108             results_formatted.append({'room_number': beacon[1], '
    description': beacon[2], 'id': str(beacon[0]), 'zone_name': zone_name})
109         print(results_formatted)
110         return {
111             'beacons': results_formatted,
112             'successful': True
113         }
114
115     def get_beacons_macos(self, query):
116         """
117             Function called by a MACOS admin to get all beacons.
118
119             :returns JSON with format
120
121                 successful: boolean
122                 beacons: [beacon dictionary object]
123         """
124         sql = "SELECT * FROM beacon"
125         beacons = self.postgres_handler.select(sql)
126         results_formatted = []
127         for beacon in beacons:
128             # We need zone names as well, all we have are the zone_ids
129             sql = "SELECT zone_name FROM zone WHERE zone_id = %s"
130             zone_id = (beacon[5],)
131             zone_name = self.postgres_handler.query(sql, zone_id)[0][0]
132             results_formatted.append(
133                 {'room_number': beacon[1], 'description': beacon[2], 'id
    ': str(beacon[0]), 'zone_name': zone_name, 'major': str(beacon[3]), '
    minor': str(beacon[4])})
134         print(results_formatted)
135         return {
136             'beacons': results_formatted,
137             'successful': True
138         }
```

```
139
140     def create_beacon(self, room_number, zone_name, description):
141         """
142             Function called by admin to create a new beacon.
    Automatically generates the major and minor values to be
143             used as the identifier of the beacon. A JSON request is
    accepted in the format:
144
145                 room_number: String
146                 description: String
147                 zone_name: String
148
149             :returns JSON with format
150
151                 successful: boolean
152                 reason: String
153         """
154         # Must generate a unique set of major and minor values
155         major_value = 0
156         # minor_value definition automatic.
157         # Uses POSTGRESQL's SERIAl property to increment value of minor
    key, with uniqueness validation.
158         # Minor value now unique, major unnecessary to change as per
    minor's 2^16 size -- impractical to fill all slots.
159
160         # zone_id is what is necessary; so, firstly, get zone_id from
    zone_name (it is unique)
161         sql = "SELECT zone_id FROM zone WHERE zone_name = (%s)"
162         zone_name = (zone_name,)
163         # Obtained zone_id
164         zone_id = self.postgres_handler.query(sql, zone_name)[0][0]
165
166         args = (room_number, description, major_value, zone_id)
167         sql = "INSERT INTO beacon (beacon_id, room_number, description,
    major_key, minor_key, zone_zone_id) VALUES (DEFAULT , %s, %s, %s,
    DEFAULT, %s)"
168
169         self.postgres_handler.insert(sql, args)
170
171         return {
172             'successful': True,
173             'reason': 'No query provided'
174         }
175
176     def edit_beacon(self, beacon_id, room_number=None, description=None,
     zone_name=None):
177         """
178             Allows an admin to edit a beacon from the database. A JSON
    request
```

```
179             is accepted in the format:
180 
181                 beacon_id: String
182                 room_number: String
183                 description: String
184                 zone_name: String
185 
186             :returns JSON with format
187 
188                 successful: boolean
189                 reason: String
190         """
191         if beacon_id:
192             # Precondition: Ensure that beacon_id is not equal to None
193             if room_number:
194                 sql = "UPDATE beacon SET room_number = '%s' WHERE
    beacon_id = '%s';" % (room_number, beacon_id)
195                 self.postgres_handler.update(sql),
196             if description:
197                 sql = "UPDATE beacon SET description = '%s' WHERE
    beacon_id = '%s';" % (description, beacon_id)
198                 self.postgres_handler.update(sql),
199             if zone_name:
200                 # zone_id is what is necessary; so, firstly, get zone_id
     from zone_name (it is unique)
201                 sql = "SELECT zone_id FROM zone WHERE zone_name = (%s)"
202                 zone_name = (zone_name,)
203                 # Obtained zone_id
204                 zone_id = self.postgres_handler.query(sql, zone_name)[0]
    [0]
205 
206                 # Use this to update the beacon
207                 sql = "UPDATE beacon SET zone_zone_id = %s WHERE
    beacon_id = %s;"
208                 args = (zone_id, beacon_id)
209                 self.postgres_handler.insert(sql, args)
210             return {
211                 'successful': True,
212                 'reason': 'Edit(s) made.'
213             }
214         return {
215             'successful': False,
216             'reason': 'No query provided'
217         }
218 
219     def delete_beacon(self, beacon_id):
220         """
221             Allows an admin to delete a beacon from the database. A JSON
     request
```

```
222             is accepted in the format:
223 
224                 beacon_id: String
225 
226             :returns JSON with format
227 
228                 successful: boolean
229                 reason: String
230         """
231         if beacon_id:
232             # Precondition: Ensure that beacon_id is not equal to None
233             sql = "DELETE FROM beacon WHERE beacon_id = %s;"
234             self.postgres_handler.query(sql, beacon_id)
235 
236             return {
237                 'successful': True,
238                 'reason': 'Beacon deleted'
239             }
240         return {
241             'successful': False,
242             'reason': 'No query provided'
243         }
244 
245     def add_admin(self, name, username, password, conf_password):
246         """
247             Allows an admin to add another admin to the database. The
    JSON request
248             accepted is in the format:
249 
250                 name: String
251                 email: String
252                 password: String
253                 conf_password: String
254 
255             :returns JSON with format
256 
257                 successful: boolean
258                 reason: String
259             """
260 
261         # Instantiating return_request to be sent to client of server
262         return_request = {
263             'successful': False,
264             'reason': 'Unknown'
265         }
266         # If passwords match
267         if password == conf_password:
268             sql = "SELECT * FROM admin WHERE LOWER(email) = LOWER(%s);"
269             _username = (username,)
```

```
270             # If the email is not in use (not in the database)
271             if not self.postgres_handler.query(sql, _username):
272                 try:
273                     # Hashing password using hashlib
274                     pw_hash = phandler.hash_password(password)
275                     # Adding new user to database: name, username,
    hashed password
276                     sql = "INSERT INTO admin (admin_id, name, password,
    email) VALUES (DEFAULT , %s, %s, %s)"
277                     args = (name, pw_hash, username)
278                     self.postgres_handler.insert(sql, args)
279 
280                     return_request['successful'] = True
281                     return_request['reason'] = 'Please login again using
     your credentials.'
282                     return return_request
283                 except:
284                     # Fail condition: Broad fail condition for failure
    to connect to database
285                     return_request['reason'] = 'con_error'
286                     return return_request
287             else:
288                 # Fail condition: Email (username) in use
289                 return_request['reason'] = 'email_use'
290                 return return_request
291         else:
292             # Fail condition: Passwords do not match
293             return_request['reason'] = 'pass_match'
294             return return_request
```