

Patryk Mroczyński
Jakub Wiśniewski
Daniel Stańczak
Oskar Rutkowski

Sauron

System Alarmujący o Uczniach Robiących Oszustwa Naukowe

“Najpiękniejsze w byciu studentem jest to, że można
zrobić projekt, mający na celu utrudnianie życia innym
studentom”

Podział prac

Patryk Mroczyński:

Aplikacja serwerowa, baza danych - Python, MongoDB

Oskar Rutkowski:

Aplikacja prowadzącego, interfejs - aplikacja webowa

Daniel Stańczak:

Aplikacja monitorująca procesy i odwiedzane strony - Python

Jakub Wiśniewski:

Aplikacja prowadzącego - aplikacja webowa



Aplikacja prowadzącego

Zaimplementowane funkcjonalności:


1. Obsługa logowania i wylogowania
2. Widok wyboru sali
3. Możliwość wyboru sali w menu
4. Możliwość wyboru etykiety w menu
5. Możliwość podglądu użytkownika z listy użytkowników w sali



Aplikacja prowadzącego - widok logowania:

Login

Zaloguj



Aplikacja prowadzącego - widok logowania (error):

Login


123

...

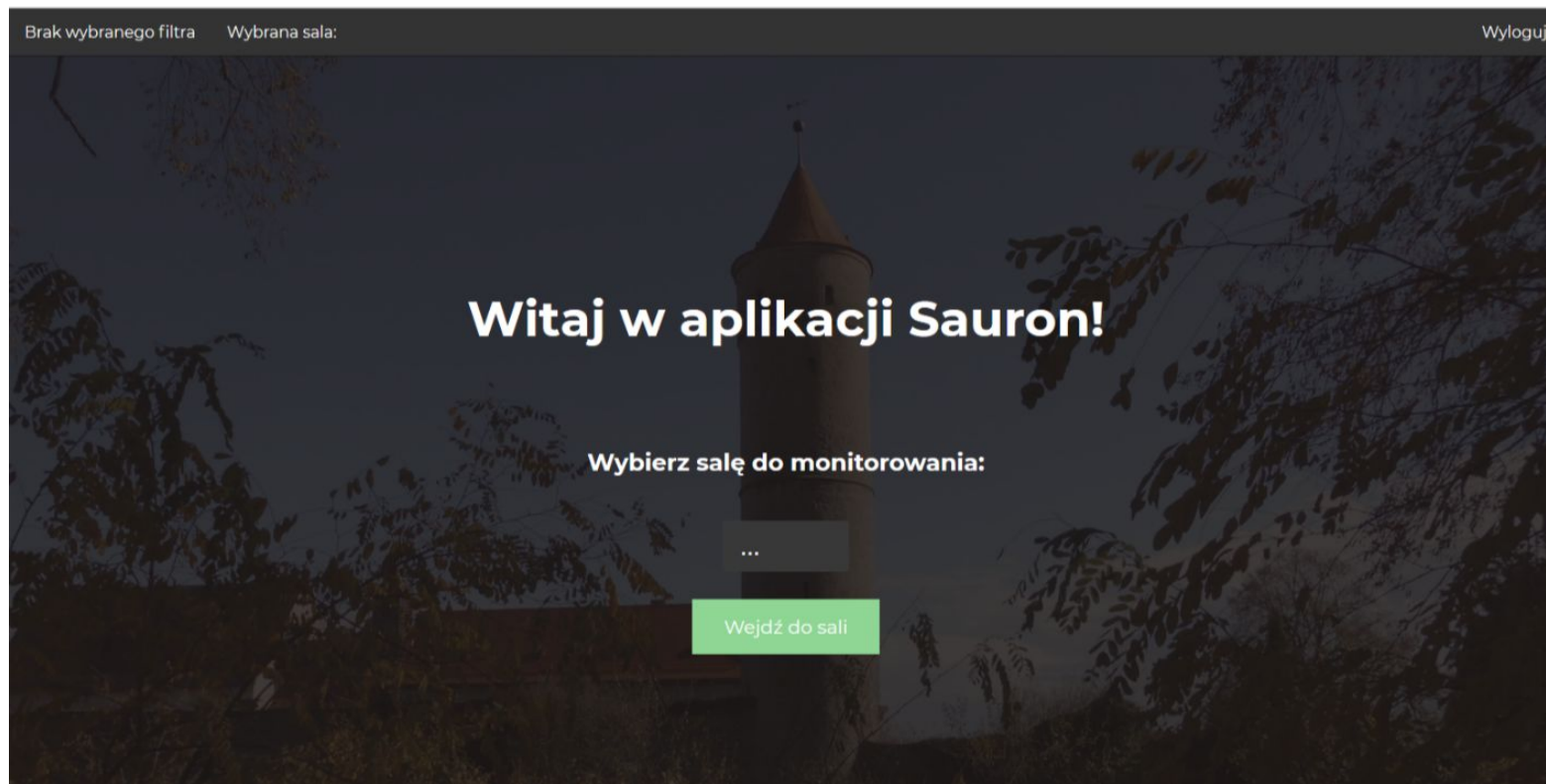
Zaloguj

Wystąpił błąd

Logowanie nie powiodło się



Aplikacja prowadzącego - widok wyboru sali:



Aplikacja prowadzącego - widok użytkowników w sali:

Brak wybranego filtraWybrana sala: 215Wyloguj

Aktualnie podłączeni
użytkownicy do sali

Nazwa:	Akcja:
Leanne Graham	<button>Podgląd</button> <button>Zbanuj</button>
Ervin Howell	<button>Podgląd</button> <button>Zbanuj</button>
Clementine Bauch	<button>Podgląd</button> <button>Zbanuj</button>
Patricia Lebsack	<button>Podgląd</button> <button>Zbanuj</button>
Chelsey Dietrich	<button>Podgląd</button> <button>Zbanuj</button>
Mrs. Dennis Schulist	<button>Podgląd</button> <button>Zbanuj</button>
Kurtis Weissnat	<button>Podgląd</button> <button>Zbanuj</button>
Nicholas Runolfsdottir V	<button>Podgląd</button> <button>Zbanuj</button>
Glenna Reichert	<button>Podgląd</button> <button>Zbanuj</button>
Clementina DuBuque	<button>Podgląd</button> <button>Zbanuj</button>

Marek Hucz

Uruchomione procesy:

-
-
-

systemd
kthread
python3

Aplikacja kliencka - <https://git.io/vpNWZ>

Zaimplementowane funkcjonalności:

1. Wczytywanie konfiguracji z pliku
2. Periodyczne zbieranie listy uruchomionych procesów
3. Wysyłanie listy zebranych procesów do serwera

Funkcjonalności do zaimplementowania:

1. Periodyczne wysyłanie zrzutów ekranu
 2. Rejestrowanie odwiedzanych stron
 3. Obsługa whitelisty
- 

Aplikacja kliencka - plik konfiguracyjny

```
1 {
2     "server": {
3         "protocol": "http",
4         "address": "127.0.0.1",
5         "port": 8080,
6         "process_endpoint": "/api/process/"
7     },
8     "name": "name",
9     "group": "group",
10    "time period": {
11        "processes collecting": 30,
12        "server communication": 60
13    }
14 }
```

Server

```
config.py x
1 import cherryypy
2 from database_management import COLLECTIONS
3
4 def validate_password(realm, login, password):
5     with COLLECTIONS['users'] as col:
6         user_exists = list(col.find({'login': login, 'password': password}))
7     return bool(user_exists)
8
9 CHERRYPY_CONFIG_DEFAULT = {
10     'server.socket_host': '10.0.2.15',
11     'server.socket_port': 8080,
12     'tools.auth_basic.on': True,
13     'tools.auth_basic.realm': '10.0.2.15',
14     'tools.auth_basic.checkpassword': validate_password,
15 }
16
17 WITHOUT_AUTHENTICATION = {'/': {'request.dispatch': cherryypy.dispatch.MethodDispatcher(), 'tools.auth_basic.on': False}}
18 WITH_AUTHENTICATION = {'/': {'request.dispatch': cherryypy.dispatch.MethodDispatcher()}}
19
```

Serwer

```
database_management.py x
1  import pymongo
2
3  class Collection:
4
5      def __init__(self, database, name):
6          self.database = database
7          self.name = name
8          self.client = pymongo.MongoClient()
9
10     def __enter__(self):
11         return self.client[self.database][self.name]
12
13     def __exit__(self, *arg):
14         self.client.close()
15
16     DB_NAME = 'sauron'
17
18     COLLECTIONS = {
19         'users': Collection(DB_NAME, 'users'),
20         'processes': Collection(DB_NAME, 'processes'),
21         'whitelists': Collection(DB_NAME, 'whitelists'),
22     }
```

Serwer

```
process_service.py x
1 import cherrypy
2 from database_management import COLLECTIONS
3 from config import CHERRYPPY_CONFIG_DEFAULT, WITHOUT_AUTHENTICATION
4
5 @cherrypy.expose
6 class ProcessService:
7
8     @cherrypy.tools.json_out()
9     def GET(self, time_from = None, time_to = None, nazgul = None, limit = 20):
10         query = {}
11         try:
12             if time_from and time_to:
13                 query.update({'create_time': {'$lt': int(time_to), '$gte': int(time_from)}})
14             if nazgul:
15                 query.update({'nazgul': str(nazgul)})
16
17             with COLLECTIONS['processes'] as col:
18                 return list(col.find(query, {'_id': False}).limit(int(limit)))
19         except ValueError:
20             raise cherrypy.HTTPError(400, 'Bad Request')
21
22     @cherrypy.tools.json_in()
23     def POST(self):
24         request = cherrypy.request.json
25
26         with COLLECTIONS['processes'] as col:
27             try:
28                 col.insert_one({
29                     'create_time': request['create_time'],
30                     'nazgul': request['nazgul'],
31                     'processes': request['processes'],
32                     'group': request['group'],
33                     'screenshot': request.get('screenshot')
34                 })
35             except (KeyError, TypeError):
36                 raise cherrypy.HTTPError(400, 'Bad Request')
37
38
39 if __name__ == '__main__':
40     cherrypy.config.update(CHERRYPPY_CONFIG_DEFAULT)
41     cherrypy.quickstart(ProcessService(), '/api/process', WITHOUT_AUTHENTICATION)
```

Server

```
user_service.py x
1  import cherrypy
2  from database_management import COLLECTIONS
3  from config import CHERRYPY_CONFIG_DEFAULT, WITHOUT_AUTHENTICATION
4  from base64 import b64encode
5
6
7  @cherrypy.expose
8  class UserService:
9
10     @cherrypy.tools.json_out()
11     def GET(self, login, password):
12         with COLLECTIONS['users'] as col:
13             user_exists = list(col.find({'login': login, 'password': password}))
14             if user_exists:
15                 auth_token = b64encode(bytes(f'{login}:{password}', 'utf-8')).decode('utf-8')
16                 return {'auth_token': auth_token}
17             else:
18                 raise cherrypy.HTTPError(404, 'User not found')
19
20     @cherrypy.tools.json_in()
21     def POST(self):
22         request = cherrypy.request.json
23         with COLLECTIONS['users'] as col:
24             try:
25                 col.insert_one({'login': request['login'], 'password': request['password']})
26             except (KeyError, TypeError):
27                 raise cherrypy.HTTPError(400, 'Bad Request')
28
29
30 if __name__ == '__main__':
31     cherrypy.config.update(CHERRYPY_CONFIG_DEFAULT)
32     cherrypy.quickstart(UserService(), '/api/user', WITHOUT_AUTHENTICATION)
33
```

Serwer

```
whitelist_service.py x
1  import cherrypy
2  from database_management import COLLECTIONS
3  from config import CHERRYPY_CONFIG_DEFAULT, WITHOUT_AUTHENTICATION
4  from base64 import b64encode
5
6
7  @cherrypy.expose
8  class WhitelistService:
9
10     @cherrypy.tools.json_out()
11     def GET(self, only_active = True):
12         try:
13             query = {'active': True} if bool(int(only_active)) else {}
14         except ValueError:
15             raise cherrypy.HTTPError(400, 'Bad Request')
16         with COLLECTIONS['whitelists'] as col:
17             return list(col.find(query, {'_id': False}))
18
19     @cherrypy.tools.json_in()
20     def POST(self):
21         request = cherrypy.request.json
22         with COLLECTIONS['whitelists'] as col:
23             try:
24                 col.insert_one({
25                     'name': request['name'],
26                     'active': False,
27                     'processes': request['processes'],
28                     'domains': request['domains'],
29                 })
30             except (KeyError, TypeError):
31                 raise cherrypy.HTTPError(400, 'Bad Request')
32
33     def PATCH(self, name, active):
34         with COLLECTIONS['whitelists'] as col:
35             try:
36                 col.update_many({'name': name}, {'$set': {'active': bool(int(active))}})
37             except ValueError:
38                 raise cherrypy.HTTPError(400, 'Bad Request')
39
40
41 if __name__ == '__main__':
42     cherrypy.config.update(CHERRYPY_CONFIG_DEFAULT)
43     cherrypy.quickstart(WhitelistService(), '/api/whitelist', WITHOUT_AUTHENTICATION)
```



Dziękujemy za uwagę!

Patryk Mroczyński
Jakub Wiśniewski
Daniel Stańczak
Oskar Rutkowski