

6a732caa81 ▾

...

Piano-LED-Visualizer / webinterface / views_api.py / <> Jump to ▾



SirLefti showing cover state in webinterface ...

History

3 contributors



1463 lines (1125 sloc) | 67.4 KB

...

```

1  from webinterface import webinterface
2  from flask import render_template, send_file, redirect, request, url_for, jsonify
3  from lib.functions import find_between, theaterChase, theaterChaseRainbow, sound_of_da_police, sca
4      rainbow, rainbowCycle, fastColorWipe, play_midi, clamp
5  import psutil
6  import threading
7  from neopixel import *
8  import webcolors as wc
9  import mido
10 from xml.dom import minidom
11 from subprocess import call
12 import subprocess
13 import datetime
14 import os
15 import math
16 from zipfile import ZipFile
17 import json
18 import ast
19 import time
20 import RPi.GPIO as GPIO
21
22 SENSECOVER = 12
23 GPIO.setmode(GPIO.BCM)
24 GPIO.setup(SENSECOVER, GPIO.IN, GPIO.PUD_UP)
25
26 @webinterface.route('/api/start_animation', methods=['GET'])
27 def start_animation():
28     choice = request.args.get('name')
29     speed = request.args.get('speed')

```

```

30     if choice == "theaterchase":
31         webinterface.menu.t = threading.Thread(target=theaterChase, args=(webinterface.ledstrip.st
32                                     Color(127, 127, 127),
33                                     webinterface.ledsettings
34                                     webinterface.menu))
35         webinterface.menu.t.start()
36
37     if choice == "theaterchaserainbow":
38         webinterface.t = threading.Thread(target=theaterChaseRainbow, args=(webinterface.ledstrip.
39                                     webinterface.ledsettin
40                                     webinterface.menu, 5))
41         webinterface.t.start()
42
43     if choice == "soundofdapolice":
44         webinterface.t = threading.Thread(target=sound_of_da_police, args=(webinterface.ledstrip.s
45                                     webinterface.ledsetting
46                                     webinterface.menu, 1))
47         webinterface.t.start()
48
49     if choice == "scanner":
50         webinterface.t = threading.Thread(target=scanner, args=(webinterface.ledstrip.strip,
51                                     webinterface.ledsettings,
52                                     webinterface.menu, 1))
53         webinterface.t.start()
54
55     if choice == "breathing":
56         if speed == "fast":
57             webinterface.t = threading.Thread(target=breathing, args=(webinterface.ledstrip.strip,
58                                     webinterface.ledsettings,
59                                     webinterface.menu, 5))
60             webinterface.t.start()
61         if speed == "medium":
62             webinterface.t = threading.Thread(target=breathing, args=(webinterface.ledstrip.strip,
63                                     webinterface.ledsettings,
64                                     webinterface.menu, 10))
65             webinterface.t.start()
66         if speed == "slow":
67             webinterface.t = threading.Thread(target=breathing, args=(webinterface.ledstrip.strip,
68                                     webinterface.ledsettings,
69                                     webinterface.menu, 25))
70             webinterface.t.start()
71
72     if choice == "rainbow":
73         if speed == "fast":
74             webinterface.t = threading.Thread(target=rainbow, args=(webinterface.ledstrip.strip,
75                                     webinterface.ledsettings,
76                                     webinterface.menu, 2))
77             webinterface.t.start()
78         if speed == "medium":

```

```

79         webinterface.t = threading.Thread(target=rainbow, args=(webinterface.ledstrip.strip,
80                                                                    webinterface.ledsettings,
81                                                                    webinterface.menu, 20))
82         webinterface.t.start()
83     if speed == "slow":
84         webinterface.t = threading.Thread(target=rainbow, args=(webinterface.ledstrip.strip,
85                                                                    webinterface.ledsettings,
86                                                                    webinterface.menu, 50))
87         webinterface.t.start()
88
89     if choice == "rainbowcycle":
90         if speed == "fast":
91             webinterface.t = threading.Thread(target=rainbowCycle, args=(webinterface.ledstrip.str
92                                                                            webinterface.ledsettings,
93                                                                            webinterface.menu, 1))
94             webinterface.t.start()
95         if speed == "medium":
96             webinterface.t = threading.Thread(target=rainbowCycle, args=(webinterface.ledstrip.str
97                                                                            webinterface.ledsettings,
98                                                                            webinterface.menu, 20))
99             webinterface.t.start()
100        if speed == "slow":
101            webinterface.t = threading.Thread(target=rainbowCycle, args=(webinterface.ledstrip.str
102                                                                           webinterface.ledsettings,
103                                                                           webinterface.menu, 50))
104            webinterface.t.start()
105
106    if choice == "stop":
107        webinterface.menu.screensaver_is_running = False
108
109    return jsonify(success=True)
110
111
112 @webinterface.route('/api/get_homepage_data')
113 def get_homepage_data():
114     try:
115         temp = find_between(str(psutil.sensors_temperatures()["cpu_thermal"]), "current=", ",")
116     except:
117         temp = find_between(str(psutil.sensors_temperatures()["cpu-thermal"]), "current=", ",")
118
119     temp = round(float(temp), 1)
120
121     upload = psutil.net_io_counters().bytes_sent
122     download = psutil.net_io_counters().bytes_recv
123
124     card_space = psutil.disk_usage('/')
125
126     cover_opened = GPIO.input(SENSECOVER)
127

```

```

128     homepage_data = {
129         'cpu_usage': psutil.cpu_percent(interval=0.1),
130         'memory_usage_percent': psutil.virtual_memory()[2],
131         'memory_usage_total': psutil.virtual_memory()[0],
132         'memory_usage_used': psutil.virtual_memory()[3],
133         'cpu_temp': temp,
134         'upload': upload,
135         'download': download,
136         'card_space_used': card_space.used,
137         'card_space_total': card_space.total,
138         'card_space_percent': card_space.percent,
139         'cover_state': 'Opened' if cover_opened else 'Closed'
140     }
141     return jsonify(homepage_data)
142
143
144 @webinterface.route('/api/change_setting', methods=['GET'])
145 def change_setting():
146     setting_name = request.args.get('setting_name')
147     value = request.args.get('value')
148     second_value = request.args.get('second_value')
149     disable_sequence = request.args.get('disable_sequence')
150
151     reload_sequence = True
152     if (second_value == "no_reload"):
153         reload_sequence = False
154
155     if (disable_sequence == "true"):
156         webinterface.ledsettings.__init__(webinterface.usersettings)
157         webinterface.ledsettings.sequence_active = False
158
159     if setting_name == "clean_ledstrip":
160         fastColorWipe(webinterface.ledstrip.strip, True, webinterface.ledsettings)
161
162     if setting_name == "led_color":
163         rgb = wc.hex_to_rgb("#" + value)
164
165         webinterface.ledsettings.color_mode = "Single"
166
167         webinterface.ledsettings.red = rgb[0]
168         webinterface.ledsettings.green = rgb[1]
169         webinterface.ledsettings.blue = rgb[2]
170
171         webinterface.usersettings.change_setting_value("color_mode", webinterface.ledsettings.color_mode)
172         webinterface.usersettings.change_setting_value("red", rgb[0])
173         webinterface.usersettings.change_setting_value("green", rgb[1])
174         webinterface.usersettings.change_setting_value("blue", rgb[2])
175
176     return jsonify(success=True, reload_sequence=reload_sequence)

```

```
177
178     if setting_name == "light_mode":
179         webinterface.ledsettings.mode = value
180         webinterface.usersettings.change_setting_value("mode", value)
181
182     if setting_name == "fading_speed" or setting_name == "velocity_speed":
183         webinterface.ledsettings.fadingspeed = int(value)
184         webinterface.usersettings.change_setting_value("fadingspeed", webinterface.ledsettings.fad
185
186     if setting_name == "brightness":
187         webinterface.usersettings.change_setting_value("brightness_percent", int(value))
188         webinterface.ledstrip.change_brightness(int(value), True)
189
190     if setting_name == "backlight_brightness":
191         webinterface.ledsettings.backlight_brightness_percent = int(value)
192         webinterface.ledsettings.backlight_brightness = 255 * webinterface.ledsettings.backlight_b
193         webinterface.usersettings.change_setting_value("backlight_brightness",
194                                                         int(webinterface.ledsettings.backlight_brig
195         webinterface.usersettings.change_setting_value("backlight_brightness_percent",
196                                                         webinterface.ledsettings.backlight_brightne
197         fastColorWipe(webinterface.ledstrip.strip, True, webinterface.ledsettings)
198
199     if setting_name == "backlight_color":
200         rgb = wc.hex_to_rgb("#" + value)
201
202         webinterface.ledsettings.backlight_red = rgb[0]
203         webinterface.ledsettings.backlight_green = rgb[1]
204         webinterface.ledsettings.backlight_blue = rgb[2]
205
206         webinterface.usersettings.change_setting_value("backlight_red", rgb[0])
207         webinterface.usersettings.change_setting_value("backlight_green", rgb[1])
208         webinterface.usersettings.change_setting_value("backlight_blue", rgb[2])
209
210         fastColorWipe(webinterface.ledstrip.strip, True, webinterface.ledsettings)
211
212     if setting_name == "sides_color":
213         rgb = wc.hex_to_rgb("#" + value)
214
215         webinterface.ledsettings.adjacent_red = rgb[0]
216         webinterface.ledsettings.adjacent_green = rgb[1]
217         webinterface.ledsettings.adjacent_blue = rgb[2]
218
219         webinterface.usersettings.change_setting_value("adjacent_red", rgb[0])
220         webinterface.usersettings.change_setting_value("adjacent_green", rgb[1])
221         webinterface.usersettings.change_setting_value("adjacent_blue", rgb[2])
222
223     if setting_name == "sides_color_mode":
224         webinterface.ledsettings.adjacent_mode = value
225         webinterface.usersettings.change_setting_value("adjacent_mode", value)
```

```
226
227     if setting_name == "input_port":
228         webinterface.usersettings.change_setting_value("input_port", value)
229         webinterface.midiports.change_port("inport", value)
230
231     if setting_name == "secondary_input_port":
232         webinterface.usersettings.change_setting_value("secondary_input_port", value)
233
234     if setting_name == "play_port":
235         webinterface.usersettings.change_setting_value("play_port", value)
236         webinterface.midiports.change_port("playport", value)
237
238     if setting_name == "skipped_notes":
239         webinterface.usersettings.change_setting_value("skipped_notes", value)
240         webinterface.ledsettings.skipped_notes = value
241
242     if setting_name == "add_note_offset":
243         webinterface.ledsettings.add_note_offset()
244         return jsonify(success=True, reload=True)
245
246     if setting_name == "append_note_offset":
247         webinterface.ledsettings.append_note_offset()
248         return jsonify(success=True, reload=True)
249
250     if setting_name == "remove_note_offset":
251         webinterface.ledsettings.del_note_offset(int(value) + 1)
252         return jsonify(success=True, reload=True)
253
254     if setting_name == "note_offsets":
255         webinterface.usersettings.change_setting_value("note_offsets", value)
256
257     if setting_name == "update_note_offset":
258         webinterface.ledsettings.update_note_offset(int(value) + 1, second_value)
259         return jsonify(success=True, reload=True)
260
261     if setting_name == "led_count":
262         webinterface.usersettings.change_setting_value("led_count", int(value))
263         webinterface.ledstrip.change_led_count(int(value), True)
264
265     if setting_name == "shift":
266         webinterface.usersettings.change_setting_value("shift", int(value))
267         webinterface.ledstrip.change_shift(int(value), True)
268
269     if setting_name == "reverse":
270         webinterface.usersettings.change_setting_value("reverse", int(value))
271         webinterface.ledstrip.change_reverse(int(value), True)
272
273     if setting_name == "color_mode":
274         reload_sequence = True
```

```
275         if (second_value == "no_reload"):
276             reload_sequence = False
277
278         webinterface.ledsettings.color_mode = value
279         webinterface.usersettings.change_setting_value("color_mode", webinterface.ledsettings.color_mode)
280         return jsonify(success=True, reload_sequence=reload_sequence)
281
282     if setting_name == "add_multicolor":
283         webinterface.ledsettings.addcolor()
284         return jsonify(success=True, reload=True)
285
286     if setting_name == "add_multicolor_and_set_value":
287         settings = json.loads(value)
288
289         webinterface.ledsettings.multicolor.clear()
290         webinterface.ledsettings.multicolor_range.clear()
291
292         for key, value in settings.items():
293             rgb = wc.hex_to_rgb("#" + value["color"])
294
295             webinterface.ledsettings.multicolor.append([int(rgb[0]), int(rgb[1]), int(rgb[2])])
296             webinterface.ledsettings.multicolor_range.append([int(value["range"][0]), int(value["range"][1])])
297
298         webinterface.usersettings.change_setting_value("multicolor", webinterface.ledsettings.multicolor)
299         webinterface.usersettings.change_setting_value("multicolor_range", webinterface.ledsettings.multicolor_range)
300
301         return jsonify(success=True)
302
303
304     if setting_name == "remove_multicolor":
305         webinterface.ledsettings.deletecolor(int(value) + 1)
306         return jsonify(success=True, reload=True)
307
308     if setting_name == "multicolor":
309         rgb = wc.hex_to_rgb("#" + value)
310         webinterface.ledsettings.multicolor[int(second_value)][0] = rgb[0]
311         webinterface.ledsettings.multicolor[int(second_value)][1] = rgb[1]
312         webinterface.ledsettings.multicolor[int(second_value)][2] = rgb[2]
313
314         webinterface.usersettings.change_setting_value("multicolor", webinterface.ledsettings.multicolor)
315
316         return jsonify(success=True, reload_sequence=reload_sequence)
317
318     if setting_name == "multicolor_range_left":
319         webinterface.ledsettings.multicolor_range[int(second_value)][0] = int(value)
320         webinterface.usersettings.change_setting_value("multicolor_range", webinterface.ledsettings.multicolor_range)
321
322         return jsonify(success=True, reload_sequence=reload_sequence)
323
```

```
324     if setting_name == "multicolor_range_right":
325         webinterface.ledsettings.multicolor_range[int(second_value)][1] = int(value)
326         webinterface.usersettings.change_setting_value("multicolor_range", webinterface.ledsetting
327
328         return jsonify(success=True, reload_sequence=reload_sequence)
329
330     if setting_name == "remove_all_multicolors":
331         webinterface.ledsettings.multicolor.clear()
332         webinterface.ledsettings.multicolor_range.clear()
333
334         webinterface.usersettings.change_setting_value("multicolor", webinterface.ledsettings.mult
335         webinterface.usersettings.change_setting_value("multicolor_range", webinterface.ledsetting
336         return jsonify(success=True)
337
338     if setting_name == "rainbow_offset":
339         webinterface.ledsettings.rainbow_offset = int(value)
340         webinterface.usersettings.change_setting_value("rainbow_offset",
341                                                         int(webinterface.ledsettings.rainbow_offset
342         return jsonify(success=True, reload_sequence=reload_sequence)
343
344     if setting_name == "rainbow_scale":
345         webinterface.ledsettings.rainbow_scale = int(value)
346         webinterface.usersettings.change_setting_value("rainbow_scale",
347                                                         int(webinterface.ledsettings.rainbow_scale)
348         return jsonify(success=True, reload_sequence=reload_sequence)
349
350     if setting_name == "rainbow_timeshift":
351         webinterface.ledsettings.rainbow_timeshift = int(value)
352         webinterface.usersettings.change_setting_value("rainbow_timeshift",
353                                                         int(webinterface.ledsettings.rainbow_timesh
354         return jsonify(success=True, reload_sequence=reload_sequence)
355
356     if setting_name == "speed_slowest_color":
357         rgb = wc.hex_to_rgb("#" + value)
358         webinterface.ledsettings.speed_slowest["red"] = rgb[0]
359         webinterface.ledsettings.speed_slowest["green"] = rgb[1]
360         webinterface.ledsettings.speed_slowest["blue"] = rgb[2]
361
362         webinterface.usersettings.change_setting_value("speed_slowest_red", rgb[0])
363         webinterface.usersettings.change_setting_value("speed_slowest_green", rgb[1])
364         webinterface.usersettings.change_setting_value("speed_slowest_blue", rgb[2])
365
366         return jsonify(success=True, reload_sequence=reload_sequence)
367
368     if setting_name == "speed_fastest_color":
369         rgb = wc.hex_to_rgb("#" + value)
370         webinterface.ledsettings.speed_fastest["red"] = rgb[0]
371         webinterface.ledsettings.speed_fastest["green"] = rgb[1]
372         webinterface.ledsettings.speed_fastest["blue"] = rgb[2]
```



```
373
374     webinterface.usersettings.change_setting_value("speed_fastest_red", rgb[0])
375     webinterface.usersettings.change_setting_value("speed_fastest_green", rgb[1])
376     webinterface.usersettings.change_setting_value("speed_fastest_blue", rgb[2])
377
378     return jsonify(success=True, reload_sequence=reload_sequence)
379
380 if setting_name == "gradient_start_color":
381     rgb = wc.hex_to_rgb("#" + value)
382     webinterface.ledsettings.gradient_start["red"] = rgb[0]
383     webinterface.ledsettings.gradient_start["green"] = rgb[1]
384     webinterface.ledsettings.gradient_start["blue"] = rgb[2]
385
386     webinterface.usersettings.change_setting_value("gradient_start_red", rgb[0])
387     webinterface.usersettings.change_setting_value("gradient_start_green", rgb[1])
388     webinterface.usersettings.change_setting_value("gradient_start_blue", rgb[2])
389
390     return jsonify(success=True, reload_sequence=reload_sequence)
391
392 if setting_name == "gradient_end_color":
393     rgb = wc.hex_to_rgb("#" + value)
394     webinterface.ledsettings.gradient_end["red"] = rgb[0]
395     webinterface.ledsettings.gradient_end["green"] = rgb[1]
396     webinterface.ledsettings.gradient_end["blue"] = rgb[2]
397
398     webinterface.usersettings.change_setting_value("gradient_end_red", rgb[0])
399     webinterface.usersettings.change_setting_value("gradient_end_green", rgb[1])
400     webinterface.usersettings.change_setting_value("gradient_end_blue", rgb[2])
401
402     return jsonify(success=True, reload_sequence=reload_sequence)
403
404 if setting_name == "speed_max_notes":
405     webinterface.ledsettings.speed_max_notes = int(value)
406     webinterface.usersettings.change_setting_value("speed_max_notes", int(value))
407
408     return jsonify(success=True, reload_sequence=reload_sequence)
409
410 if setting_name == "speed_period_in_seconds":
411     webinterface.ledsettings.speed_period_in_seconds = float(value)
412     webinterface.usersettings.change_setting_value("speed_period_in_seconds", float(value))
413
414     return jsonify(success=True, reload_sequence=reload_sequence)
415
416 if setting_name == "key_in_scale_color":
417     rgb = wc.hex_to_rgb("#" + value)
418     webinterface.ledsettings.key_in_scale["red"] = rgb[0]
419     webinterface.ledsettings.key_in_scale["green"] = rgb[1]
420     webinterface.ledsettings.key_in_scale["blue"] = rgb[2]
421
```

```
422     webinterface.usersettings.change_setting_value("key_in_scale_red", rgb[0])
423     webinterface.usersettings.change_setting_value("key_in_scale_green", rgb[1])
424     webinterface.usersettings.change_setting_value("key_in_scale_blue", rgb[2])
425
426     return jsonify(success=True, reload_sequence=reload_sequence)
427
428     if setting_name == "key_not_in_scale_color":
429         rgb = wc.hex_to_rgb("#" + value)
430         webinterface.ledsettings.key_not_in_scale["red"] = rgb[0]
431         webinterface.ledsettings.key_not_in_scale["green"] = rgb[1]
432         webinterface.ledsettings.key_not_in_scale["blue"] = rgb[2]
433
434         webinterface.usersettings.change_setting_value("key_not_in_scale_red", rgb[0])
435         webinterface.usersettings.change_setting_value("key_not_in_scale_green", rgb[1])
436         webinterface.usersettings.change_setting_value("key_not_in_scale_blue", rgb[2])
437
438         return jsonify(success=True, reload_sequence=reload_sequence)
439
440     if setting_name == "scale_key":
441         webinterface.ledsettings.scale_key = int(value)
442         webinterface.usersettings.change_setting_value("scale_key", int(value))
443
444         return jsonify(success=True, reload_sequence=reload_sequence)
445
446     if setting_name == "next_step":
447         webinterface.ledsettings.set_sequence(0, 1, False)
448         return jsonify(success=True, reload_sequence=reload_sequence)
449
450     if setting_name == "set_sequence":
451         if (int(value) == 0):
452             webinterface.ledsettings.__init__(webinterface.usersettings)
453             webinterface.ledsettings.sequence_active = False
454         else:
455             webinterface.ledsettings.set_sequence(int(value) - 1, 0)
456         return jsonify(success=True, reload_sequence=reload_sequence)
457
458     if setting_name == "change_sequence_name":
459         sequences_tree = minidom.parse("sequences.xml")
460         sequence_to_edit = "sequence_" + str(value)
461
462         sequences_tree.getElementsByTagName(sequence_to_edit)[
463             0].getElementsByTagName("settings")[
464             0].getElementsByTagName("sequence_name")[0].firstChild.nodeValue = str(second_value)
465
466         pretty_save("sequences.xml", sequences_tree)
467
468         return jsonify(success=True, reload_sequence=reload_sequence)
469
470     if setting_name == "change_step_value":
```

```
471     sequences_tree = minidom.parse("sequences.xml")
472     sequence_to_edit = "sequence_" + str(value)
473
474     sequences_tree.getElementsByTagName(sequence_to_edit)[
475         0].getElementsByTagName("settings")[
476         0].getElementsByTagName("next_step")[0].firstChild.nodeValue = str(second_value)
477
478     pretty_save("sequences.xml", sequences_tree)
479
480     return jsonify(success=True, reload_sequence=reload_sequence)
481
482 if setting_name == "change_step_activation_method":
483     sequences_tree = minidom.parse("sequences.xml")
484     sequence_to_edit = "sequence_" + str(value)
485
486     sequences_tree.getElementsByTagName(sequence_to_edit)[
487         0].getElementsByTagName("settings")[
488         0].getElementsByTagName("control_number")[0].firstChild.nodeValue = str(second_value)
489
490     pretty_save("sequences.xml", sequences_tree)
491
492     return jsonify(success=True, reload_sequence=reload_sequence)
493
494 if setting_name == "add_sequence":
495     sequences_tree = minidom.parse("sequences.xml")
496
497     sequences_amount = 1
498     while True:
499         if (len(sequences_tree.getElementsByTagName("sequence_" + str(sequences_amount)))) == 0
500             break
501         sequences_amount += 1
502
503     settings = sequences_tree.createElement("settings")
504
505     control_number = sequences_tree.createElement("control_number")
506     control_number.appendChild(sequences_tree.createTextNode("0"))
507     settings.appendChild(control_number)
508
509     next_step = sequences_tree.createElement("next_step")
510     next_step.appendChild(sequences_tree.createTextNode("1"))
511     settings.appendChild(next_step)
512
513     sequence_name = sequences_tree.createElement("sequence_name")
514     sequence_name.appendChild(sequences_tree.createTextNode("Sequence " + str(sequences_amount)
515     settings.appendChild(sequence_name)
516
517     step = sequences_tree.createElement("step_1")
518
519     color = sequences_tree.createElement("color")
```

```

520     color.appendChild(sequences_tree.createTextNode("RGB"))
521     step.appendChild(color)
522
523     red = sequences_tree.createElement("Red")
524     red.appendChild(sequences_tree.createTextNode("255"))
525     step.appendChild(red)
526
527     green = sequences_tree.createElement("Green")
528     green.appendChild(sequences_tree.createTextNode("255"))
529     step.appendChild(green)
530
531     blue = sequences_tree.createElement("Blue")
532     blue.appendChild(sequences_tree.createTextNode("255"))
533     step.appendChild(blue)
534
535     light_mode = sequences_tree.createElement("light_mode")
536     light_mode.appendChild(sequences_tree.createTextNode("Normal"))
537     step.appendChild(light_mode)
538
539     element = sequences_tree.createElement("sequence_" + str(sequences_amount))
540     element.appendChild(settings)
541     element.appendChild(step)
542
543     sequences_tree.getElementsByTagName("list")[0].appendChild(element)
544
545     pretty_save("sequences.xml", sequences_tree)
546
547     return jsonify(success=True, reload_sequence=reload_sequence)
548
549 if setting_name == "remove_sequence":
550     sequences_tree = minidom.parse("sequences.xml")
551
552     # removing sequence node
553     nodes = sequences_tree.getElementsByTagName("sequence_" + str(value))
554     for node in nodes:
555         parent = node.parentNode
556         parent.removeChild(node)
557
558     # changing nodes tag names
559     i = 1
560     for sequence in sequences_tree.getElementsByTagName("list")[0].childNodes:
561         if (sequence.nodeType == 1):
562             sequences_tree.getElementsByTagName(sequence.nodeName)[0].tagName = "sequence_" +
563             i += 1
564
565     pretty_save("sequences.xml", sequences_tree)
566
567     return jsonify(success=True, reload_sequence=reload_sequence)
568

```

```

569     if setting_name == "add_step":
570         sequences_tree = minidom.parse("sequences.xml")
571
572         step_amount = 1
573         while True:
574             if (len(sequences_tree.getElementsByTagName("sequence_" + str(value))[0].getElementsByTagName(
575                 "step_" + str(step_amount))) == 0):
576                 break
577             step_amount += 1
578
579         step = sequences_tree.createElement("step_" + str(step_amount))
580
581         color = sequences_tree.createElement("color")
582
583         color.appendChild(sequences_tree.createTextNode("RGB"))
584         step.appendChild(color)
585
586         red = sequences_tree.createElement("Red")
587         red.appendChild(sequences_tree.createTextNode("255"))
588         step.appendChild(red)
589
590         green = sequences_tree.createElement("Green")
591         green.appendChild(sequences_tree.createTextNode("255"))
592         step.appendChild(green)
593
594         blue = sequences_tree.createElement("Blue")
595         blue.appendChild(sequences_tree.createTextNode("255"))
596         step.appendChild(blue)
597
598         light_mode = sequences_tree.createElement("light_mode")
599         light_mode.appendChild(sequences_tree.createTextNode("Normal"))
600         step.appendChild(light_mode)
601
602         sequences_tree.getElementsByTagName("sequence_" + str(value))[0].appendChild(step)
603
604         pretty_save("sequences.xml", sequences_tree)
605
606         return jsonify(success=True, reload_sequence=reload_sequence, reload_steps_list=True)
607
608     # remove node list with a tag name "step_" + str(value), and change tag names to maintain orde
609     if setting_name == "remove_step":
610
611         second_value = int(second_value)
612         second_value += 1
613
614         sequences_tree = minidom.parse("sequences.xml")
615
616         # removing step node
617         nodes = sequences_tree.getElementsByTagName("sequence_" + str(value))[0].getElementsByTagNameN

```

```

618         "step_" + str(second_value))
619     for node in nodes:
620         parent = node.parentNode
621         parent.removeChild(node)
622
623     # changing nodes tag names
624     i = 1
625     for step in sequences_tree.getElementsByTagName("sequence_" + str(value))[0].childNodes:
626         if (step.nodeType == 1 and step.tagName != "settings"):
627             sequences_tree.getElementsByTagName("sequence_" + str(value))[0].getElementsByTagName
628                 [0].tagName = "step_" + str(i)
629             i += 1
630
631     pretty_save("sequences.xml", sequences_tree)
632
633     return jsonify(success=True, reload_sequence=reload_sequence)
634
635     # saving current led settings as sequence step
636     if setting_name == "save_led_settings_to_step" and second_value != "":
637
638         # remove node and child under "sequence_" + str(value) and "step_" + str(second_value)
639         sequences_tree = minidom.parse("sequences.xml")
640
641         second_value = int(second_value)
642         second_value += 1
643
644         nodes = sequences_tree.getElementsByTagName("sequence_" + str(value))[0].getElementsByTagName
645             ("step_" + str(second_value))
646         for node in nodes:
647             parent = node.parentNode
648             parent.removeChild(node)
649
650         # create new step node
651         step = sequences_tree.createElement("step_" + str(second_value))
652
653         # load color mode from webinterface.ledsettings and put it into step node
654         color_mode = sequences_tree.createElement("color")
655         color_mode.appendChild(sequences_tree.createTextNode(str(webinterface.ledsettings.color_mo
656         step.appendChild(color_mode)
657
658         # load mode from webinterface.ledsettings and put it into step node
659         mode = sequences_tree.createElement("light_mode")
660         mode.appendChild(sequences_tree.createTextNode(str(webinterface.ledsettings.mode)))
661         step.appendChild(mode)
662
663         # if mode is equal "Fading" or "Velocity" load mode from webinterface.ledsettings and put
664         if (webinterface.ledsettings.mode == "Fading" or webinterface.ledsettings.mode == "Velocit
665             fadingspeed = sequences_tree.createElement("fadingspeed")
666

```

```

667         # depending on fadingspeed name set different fadingspeed value
668         if (webinterface.ledsettings.fadingspeed == "Slow"):
669             fadingspeed.appendChild(sequences_tree.createTextNode("10"))
670         elif (webinterface.ledsettings.fadingspeed == "Medium"):
671             fadingspeed.appendChild(sequences_tree.createTextNode("20"))
672         elif (webinterface.ledsettings.fadingspeed == "Fast"):
673             fadingspeed.appendChild(sequences_tree.createTextNode("40"))
674         elif (webinterface.ledsettings.fadingspeed == "Very fast"):
675             fadingspeed.appendChild(sequences_tree.createTextNode("50"))
676         elif (webinterface.ledsettings.fadingspeed == "Instant"):
677             fadingspeed.appendChild(sequences_tree.createTextNode("1000"))
678         elif (webinterface.ledsettings.fadingspeed == "Very slow"):
679             fadingspeed.appendChild(sequences_tree.createTextNode("2"))
680
681         step.appendChild(fadingspeed)
682
683     # if color_mode is equal to "Single" load color from webinterface.ledsettings and put it i
684     if (webinterface.ledsettings.color_mode == "Single"):
685         red = sequences_tree.createElement("Red")
686         red.appendChild(sequences_tree.createTextNode(str(webinterface.ledsettings.red)))
687         step.appendChild(red)
688
689         green = sequences_tree.createElement("Green")
690         green.appendChild(sequences_tree.createTextNode(str(webinterface.ledsettings.green)))
691         step.appendChild(green)
692
693         blue = sequences_tree.createElement("Blue")
694         blue.appendChild(sequences_tree.createTextNode(str(webinterface.ledsettings.blue)))
695         step.appendChild(blue)
696
697     # if color_mode is equal to "Multicolor" load colors from webinterface.ledsettings and put
698     if (webinterface.ledsettings.color_mode == "Multicolor"):
699         # load value from webinterface.ledsettings.multicolor
700         multicolor = webinterface.ledsettings.multicolor
701
702         # loop through multicolor object and add each color to step node under "sequence_"+str
703         for i in range(len(multicolor)):
704             color = sequences_tree.createElement("color_" + str(i + 1))
705             new_multicolor = str(multicolor[i])
706             new_multicolor = new_multicolor.replace("[", "")
707             new_multicolor = new_multicolor.replace("]", "")
708
709             color.appendChild(sequences_tree.createTextNode(new_multicolor))
710             step.appendChild(color)
711
712         # same as above but with multicolor_range and "color_range_"+str(i)
713         multicolor_range = webinterface.ledsettings.multicolor_range
714         for i in range(len(multicolor_range)):
715             color_range = sequences_tree.createElement("color_range_" + str(i + 1))

```

```

716         new_multicolor_range = str(multicolor_range[i])
717
718         new_multicolor_range = new_multicolor_range.replace("[", "")
719         new_multicolor_range = new_multicolor_range.replace("]", "")
720         color_range.appendChild(sequences_tree.createTextNode(new_multicolor_range))
721         step.appendChild(color_range)
722
723     # if color_mode is equal to "Rainbow" load colors from webinterface.ledsettings and put it
724     if (webinterface.ledsettings.color_mode == "Rainbow"):
725         # load values rainbow_offset, rainbow_scale and rainbow_timeshift from webinterface.ledsettings
726         rainbow_offset = sequences_tree.createElement("Offset")
727         rainbow_offset.appendChild(sequences_tree.createTextNode(str(webinterface.ledsettings.rainbow_offset)))
728         step.appendChild(rainbow_offset)
729
730         rainbow_scale = sequences_tree.createElement("Scale")
731         rainbow_scale.appendChild(sequences_tree.createTextNode(str(webinterface.ledsettings.rainbow_scale)))
732         step.appendChild(rainbow_scale)
733
734         rainbow_timeshift = sequences_tree.createElement("Timeshift")
735         rainbow_timeshift.appendChild(
736             sequences_tree.createTextNode(str(webinterface.ledsettings.rainbow_timeshift)))
737         step.appendChild(rainbow_timeshift)
738
739     # if color_mode is equal to "Speed" load colors from webinterface.ledsettings and put it in
740     if (webinterface.ledsettings.color_mode == "Speed"):
741         # load values speed_slowest["red"] etc from webinterface.ledsettings and put them under
742         speed_slowest_red = sequences_tree.createElement("speed_slowest_red")
743         speed_slowest_red.appendChild(
744             sequences_tree.createTextNode(str(webinterface.ledsettings.speed_slowest["red"])))
745         step.appendChild(speed_slowest_red)
746
747         speed_slowest_green = sequences_tree.createElement("speed_slowest_green")
748         speed_slowest_green.appendChild(
749             sequences_tree.createTextNode(str(webinterface.ledsettings.speed_slowest["green"])))
750         step.appendChild(speed_slowest_green)
751
752         speed_slowest_blue = sequences_tree.createElement("speed_slowest_blue")
753         speed_slowest_blue.appendChild(
754             sequences_tree.createTextNode(str(webinterface.ledsettings.speed_slowest["blue"])))
755         step.appendChild(speed_slowest_blue)
756
757         # same as above but with "fastest"
758         speed_fastest_red = sequences_tree.createElement("speed_fastest_red")
759         speed_fastest_red.appendChild(
760             sequences_tree.createTextNode(str(webinterface.ledsettings.speed_fastest["red"])))
761         step.appendChild(speed_fastest_red)
762
763         speed_fastest_green = sequences_tree.createElement("speed_fastest_green")
764         speed_fastest_green.appendChild(

```



```

765         sequences_tree.createTextNode(str(webinterface.ledsettings.speed_fastest["green"]))
766     step.appendChild(speed_fastest_green)
767
768     speed_fastest_blue = sequences_tree.createElement("speed_fastest_blue")
769     speed_fastest_blue.appendChild(
770         sequences_tree.createTextNode(str(webinterface.ledsettings.speed_fastest["blue"]))
771     step.appendChild(speed_fastest_blue)
772
773     # load "speed_max_notes" and "speed_period_in_seconds" values from webinterface.ledset
774     # and put them under speed_max_notes and speed_period_in_seconds
775
776     speed_max_notes = sequences_tree.createElement("speed_max_notes")
777     speed_max_notes.appendChild(sequences_tree.createTextNode(str(webinterface.ledsettings
778     step.appendChild(speed_max_notes)
779
780     speed_period_in_seconds = sequences_tree.createElement("speed_period_in_seconds")
781     speed_period_in_seconds.appendChild(
782         sequences_tree.createTextNode(str(webinterface.ledsettings.speed_period_in_seconds
783     step.appendChild(speed_period_in_seconds)
784
785     # if color_mode is equal to "Gradient" load colors from webinterface.ledsettings and put i
786     if (webinterface.ledsettings.color_mode == "Gradient"):
787         # load values gradient_start_red etc from webinterface.ledsettings and put them under
788         gradient_start_red = sequences_tree.createElement("gradient_start_red")
789         gradient_start_red.appendChild(
790             sequences_tree.createTextNode(str(webinterface.ledsettings.gradient_start["red"]))
791         step.appendChild(gradient_start_red)
792
793         gradient_start_green = sequences_tree.createElement("gradient_start_green")
794         gradient_start_green.appendChild(
795             sequences_tree.createTextNode(str(webinterface.ledsettings.gradient_start["green"]
796         step.appendChild(gradient_start_green)
797
798         gradient_start_blue = sequences_tree.createElement("gradient_start_blue")
799         gradient_start_blue.appendChild(
800             sequences_tree.createTextNode(str(webinterface.ledsettings.gradient_start["blue"]
801         step.appendChild(gradient_start_blue)
802
803         # same as above but with gradient_end
804         gradient_end_red = sequences_tree.createElement("gradient_end_red")
805         gradient_end_red.appendChild(
806             sequences_tree.createTextNode(str(webinterface.ledsettings.gradient_end["red"])))
807         step.appendChild(gradient_end_red)
808
809         gradient_end_green = sequences_tree.createElement("gradient_end_green")
810         gradient_end_green.appendChild(
811             sequences_tree.createTextNode(str(webinterface.ledsettings.gradient_end["green"]
812         step.appendChild(gradient_end_green)
813

```

```

814         gradient_end_blue = sequences_tree.createElement("gradient_end_blue")
815         gradient_end_blue.appendChild(
816             sequences_tree.createTextNode(str(webinterface.ledsettings.gradient_end["blue"])))
817         step.appendChild(gradient_end_blue)
818
819     # if color_mode is equal to "Scale" load colors from webinterface.ledsettings and put it i
820     if (webinterface.ledsettings.color_mode == "Scale"):
821         # load values key_in_scale_red etc from webinterface.ledsettings and put them under ke
822         key_in_scale_red = sequences_tree.createElement("key_in_scale_red")
823         key_in_scale_red.appendChild(
824             sequences_tree.createTextNode(str(webinterface.ledsettings.key_in_scale["red"])))
825         step.appendChild(key_in_scale_red)
826
827         key_in_scale_green = sequences_tree.createElement("key_in_scale_green")
828         key_in_scale_green.appendChild(
829             sequences_tree.createTextNode(str(webinterface.ledsettings.key_in_scale["green"])))
830         step.appendChild(key_in_scale_green)
831
832         key_in_scale_blue = sequences_tree.createElement("key_in_scale_blue")
833         key_in_scale_blue.appendChild(
834             sequences_tree.createTextNode(str(webinterface.ledsettings.key_in_scale["blue"])))
835         step.appendChild(key_in_scale_blue)
836
837         # same as above but with key_not_in_scale
838         key_not_in_scale_red = sequences_tree.createElement("key_not_in_scale_red")
839         key_not_in_scale_red.appendChild(
840             sequences_tree.createTextNode(str(webinterface.ledsettings.key_not_in_scale["red"]
841         step.appendChild(key_not_in_scale_red)
842
843         key_not_in_scale_green = sequences_tree.createElement("key_not_in_scale_green")
844         key_not_in_scale_green.appendChild(
845             sequences_tree.createTextNode(str(webinterface.ledsettings.key_not_in_scale["green"]
846         step.appendChild(key_not_in_scale_green)
847
848         key_not_in_scale_blue = sequences_tree.createElement("key_not_in_scale_blue")
849         key_not_in_scale_blue.appendChild(
850             sequences_tree.createTextNode(str(webinterface.ledsettings.key_not_in_scale["blue"]
851         step.appendChild(key_not_in_scale_blue)
852
853     try:
854         sequences_tree.getElementsByTagName("sequence_" + str(value))[
855             0].insertBefore(step,
856                 sequences_tree.getElementsByTagName("sequence_" + str(value))[
857                     0].getElementsByTagName("step_" + str(second_value + 1))[0])
858     except:
859         sequences_tree.getElementsByTagName("sequence_" + str(value))[0].appendChild(step)
860
861     pretty_save("sequences.xml", sequences_tree)
862

```

```
863         return jsonify(success=True, reload_sequence=reload_sequence, reload_steps_list=True)
864
865     if setting_name == "screen_on":
866         if (int(value) == 0):
867             webinterface.menu.disable_screen()
868         else:
869             webinterface.menu.enable_screen()
870
871     if setting_name == "reset_to_default":
872         webinterface.usersettings.reset_to_default()
873
874     if setting_name == "restart_rpi":
875         call("sudo /sbin/reboot now", shell=True)
876
877     if setting_name == "turnoff_rpi":
878         call("sudo /sbin/shutdown -h now", shell=True)
879
880     if setting_name == "update_rpi":
881         call("sudo git reset --hard HEAD", shell=True)
882         call("sudo git checkout .", shell=True)
883         call("sudo git clean -fdx", shell=True)
884         call("sudo git pull origin master", shell=True)
885
886     if setting_name == "connect_ports":
887         webinterface.midiports.connectall()
888         return jsonify(success=True, reload_ports=True)
889
890     if setting_name == "disconnect_ports":
891         call("sudo aconnect -x", shell=True)
892         return jsonify(success=True, reload_ports=True)
893
894     if setting_name == "restart_rtp":
895         call("sudo systemctl restart rtpmidid", shell=True)
896
897     if setting_name == "start_recording":
898         webinterface.saving.start_recording()
899         return jsonify(success=True, reload_songs=True)
900
901     if setting_name == "cancel_recording":
902         webinterface.saving.cancel_recording()
903         return jsonify(success=True, reload_songs=True)
904
905     if setting_name == "save_recording":
906         now = datetime.datetime.now()
907         current_date = now.strftime("%Y-%m-%d %H:%M")
908         webinterface.saving.save(current_date)
909         return jsonify(success=True, reload_songs=True)
910
911     if setting_name == "change_song_name":
```

```

912         if os.path.exists("Songs/" + second_value):
913             return jsonify(success=False, reload_songs=True, error=second_value + " already exists")
914
915         if "_main" in value:
916             search_name = value.replace("_main.mid", "")
917             for fname in os.listdir('Songs'):
918                 if search_name in fname:
919                     new_name = second_value.replace(".mid", "") + fname.replace(search_name, "")
920                     os.rename('Songs/' + fname, 'Songs/' + new_name)
921
922             else:
923                 os.rename('Songs/' + value, 'Songs/' + second_value)
924                 os.rename('Songs/cache/' + value + ".p", 'Songs/cache/' + second_value + ".p")
925
926
927         return jsonify(success=True, reload_songs=True)
928
929     if setting_name == "remove_song":
930         if "_main" in value:
931             name_no_suffix = value.replace("_main.mid", "")
932             for fname in os.listdir('Songs'):
933                 if name_no_suffix in fname:
934                     os.remove("Songs/" + fname)
935
936             else:
937                 os.remove("Songs/" + value)
938
939             file_types = [".musicxml", ".xml", ".mxl", ".abc"]
940             for file_type in file_types:
941                 try:
942                     os.remove("Songs/" + value.replace(".mid", file_type))
943                 except:
944                     pass
945
946             try:
947                 os.remove("Songs/cache/" + value + ".p")
948             except:
949                 print("No cache file for " + value)
950
951         return jsonify(success=True, reload_songs=True)
952
953     if setting_name == "download_song":
954         if "_main" in value:
955             zipObj = ZipFile("Songs/" + value.replace(".mid", "") + ".zip", 'w')
956             name_no_suffix = value.replace("_main.mid", "")
957             songs_count = 0
958             for fname in os.listdir('Songs'):
959                 if name_no_suffix in fname and ".zip" not in fname:
960                     songs_count += 1
961                     zipObj.write("Songs/" + fname)

```

```

961         zipObj.close()
962     if songs_count == 1:
963         os.remove("Songs/" + value.replace(".mid", "") + ".zip")
964         return send_file("../Songs/" + value, mimetype='application/x-csv', attachment_filename=
965             as_attachment=True)
966     else:
967         return send_file("../Songs/" + value.replace(".mid", "") + ".zip", mimetype='appli
968             attachment_filename=value.replace(".mid", "") + ".zip", as_attach
969 else:
970     return send_file("../Songs/" + value, mimetype='application/x-csv', attachment_filename=
971         as_attachment=True)
972
973 if setting_name == "download_sheet_music":
974     file_types = [".musicxml", ".xml", ".mxl", ".abc"]
975     i = 0
976     while i < len(file_types):
977         try:
978             new_name = value.replace(".mid", file_types[i])
979             return send_file("../Songs/" + new_name, mimetype='application/x-csv', attachment_
980                 as_attachment=True)
981         except:
982             i += 1
983     webinterface.learning.convert_midi_to_abc(value)
984     try:
985         return send_file("../Songs/" + value.replace(".mid", ".abc"), mimetype='application/x-
986             attachment_filename=value.replace(".mid", ".abc"), as_attachment=True)
987     except:
988         print("Converting failed")
989
990
991 if setting_name == "start_midi_play":
992     webinterface.saving.t = threading.Thread(target=play_midi, args=(value, webinterface.midip
993         webinterface.saving, webi
994         webinterface.ledsettings,
995         webinterface.ledstrip))
996     webinterface.saving.t.start()
997
998     return jsonify(success=True, reload_songs=True)
999
1000 if setting_name == "stop_midi_play":
1001     webinterface.saving.is_playing_midi.clear()
1002     fastColorWipe(webinterface.ledstrip.strip, True, webinterface.ledsettings)
1003
1004     return jsonify(success=True, reload_songs=True)
1005
1006 if setting_name == "learning_load_song":
1007     webinterface.learning.t = threading.Thread(target=webinterface.learning.load_midi, args=(v
1008     webinterface.learning.t.start()
1009

```

```
1010         return jsonify(success=True, reload_learning_settings=True)
1011
1012     if setting_name == "start_learning_song":
1013         webinterface.learning.t = threading.Thread(target=webinterface.learning.learn_midi)
1014         webinterface.learning.t.start()
1015
1016         return jsonify(success=True)
1017
1018     if setting_name == "stop_learning_song":
1019         webinterface.learning.is_started_midi = False
1020         fastColorWipe(webinterface.ledstrip.strip, True, webinterface.ledsettings)
1021
1022         return jsonify(success=True)
1023
1024     if setting_name == "change_practice":
1025         value = int(value)
1026         webinterface.learning.practice = value
1027         webinterface.learning.practice = clamp(webinterface.learning.practice, 0, len(webinterface
1028         webinterface.usersettings.change_setting_value("practice", webinterface.learning.practice)
1029
1030         return jsonify(success=True)
1031
1032     if setting_name == "change_tempo":
1033         value = int(value)
1034         webinterface.learning.set_tempo = value
1035         webinterface.learning.set_tempo = clamp(webinterface.learning.set_tempo, 10, 200)
1036         webinterface.usersettings.change_setting_value("set_tempo", webinterface.learning.set_temp
1037
1038         return jsonify(success=True)
1039
1040     if setting_name == "change_hands":
1041         value = int(value)
1042         webinterface.learning.hands = value
1043         webinterface.learning.hands = clamp(webinterface.learning.hands, 0, len(webinterface.learn
1044         webinterface.usersettings.change_setting_value("hands", webinterface.learning.hands)
1045
1046         return jsonify(success=True)
1047
1048     if setting_name == "change_mute_hand":
1049         value = int(value)
1050         webinterface.learning.mute_hand = value
1051         webinterface.learning.mute_hand = clamp(webinterface.learning.mute_hand, 0, len(webinterfa
1052         webinterface.usersettings.change_setting_value("mute_hand", webinterface.learning.mute_han
1053
1054         return jsonify(success=True)
1055
1056     if setting_name == "learning_start_point":
1057         value = int(value)
1058         webinterface.learning.start_point = value
```

```
1059     webinterface.learning.start_point = clamp(webinterface.learning.start_point, 0, webinterfa
1060     webinterface.usersettings.change_setting_value("start_point", webinterface.learning.start_
1061     webinterface.learning.restart_learning()
1062
1063     return jsonify(success=True)
1064
1065 if setting_name == "learning_end_point":
1066     value = int(value)
1067     webinterface.learning.end_point = value
1068     webinterface.learning.end_point = clamp(webinterface.learning.end_point, webinterface.lear
1069     webinterface.usersettings.change_setting_value("end_point", webinterface.learning.end_poi
1070     webinterface.learning.restart_learning()
1071
1072     return jsonify(success=True)
1073
1074 if setting_name == "set_current_time_as_start_point":
1075     webinterface.learning.start_point = round(float(webinterface.learning.current_idx * 100 / f
1076     webinterface.learning.start_point = clamp(webinterface.learning.start_point, 0, webinterfa
1077     webinterface.usersettings.change_setting_value("start_point", webinterface.learning.start_
1078     webinterface.learning.restart_learning()
1079
1080     return jsonify(success=True, reload_learning_settings=True)
1081
1082 if setting_name == "set_current_time_as_end_point":
1083     webinterface.learning.end_point = round(float(webinterface.learning.current_idx * 100 / f
1084     webinterface.learning.end_point = clamp(webinterface.learning.end_point, webinterface.lear
1085     webinterface.usersettings.change_setting_value("end_point", webinterface.learning.end_poi
1086     webinterface.learning.restart_learning()
1087
1088     return jsonify(success=True, reload_learning_settings=True)
1089
1090 if setting_name == "change_handL_color":
1091     value = int(value)
1092     webinterface.learning.hand_colorL += value
1093     webinterface.learning.hand_colorL = clamp(webinterface.learning.hand_colorL, 0, len(webint
1094     webinterface.usersettings.change_setting_value("hand_colorL", webinterface.learning.hand_c
1095
1096     return jsonify(success=True, reload_learning_settings=True)
1097
1098 if setting_name == "change_handR_color":
1099     value = int(value)
1100     webinterface.learning.hand_colorR += value
1101     webinterface.learning.hand_colorR = clamp(webinterface.learning.hand_colorR, 0, len(webint
1102     webinterface.usersettings.change_setting_value("hand_colorR", webinterface.learning.hand_c
1103
1104     return jsonify(success=True, reload_learning_settings=True)
1105
1106 if setting_name == "change_learning_loop":
1107     value = int(value == 'true')
```

```

1108     webinterface.learning.is_loop_active = value
1109     webinterface.usersettings.change_setting_value("is_loop_active", webinterface.learning.is_
1110
1111     return jsonify(success=True)
1112
1113
1114     return jsonify(success=True)
1115
1116
1117 @webinterface.route('/api/get_sequence_setting', methods=['GET'])
1118 def get_sequence_setting():
1119     response = {}
1120
1121     color_mode = webinterface.ledsettings.color_mode
1122
1123     light_mode = webinterface.ledsettings.mode
1124
1125     fading_speed = webinterface.ledsettings.fadingspeed
1126
1127     red = webinterface.ledsettings.red
1128     green = webinterface.ledsettings.green
1129     blue = webinterface.ledsettings.blue
1130     led_color = wc.rgb_to_hex((int(red), int(green), int(blue)))
1131
1132     multicolor = webinterface.ledsettings.multicolor
1133     multicolor_range = webinterface.ledsettings.multicolor_range
1134
1135     rainbow_scale = webinterface.ledsettings.rainbow_scale
1136     rainbow_offset = webinterface.ledsettings.rainbow_offset
1137     rainbow_timeshift = webinterface.ledsettings.rainbow_timeshift
1138
1139     speed_slowest_red = webinterface.ledsettings.speed_slowest["red"]
1140     speed_slowest_green = webinterface.ledsettings.speed_slowest["green"]
1141     speed_slowest_blue = webinterface.ledsettings.speed_slowest["blue"]
1142     speed_slowest_color = wc.rgb_to_hex((int(speed_slowest_red), int(speed_slowest_green), int(spe
1143     response["speed_slowest_color"] = speed_slowest_color
1144
1145     speed_fastest_red = webinterface.ledsettings.speed_fastest["red"]
1146     speed_fastest_green = webinterface.ledsettings.speed_fastest["green"]
1147     speed_fastest_blue = webinterface.ledsettings.speed_fastest["blue"]
1148     speed_fastest_color = wc.rgb_to_hex((int(speed_fastest_red), int(speed_fastest_green), int(spe
1149     response["speed_fastest_color"] = speed_fastest_color
1150
1151     gradient_start_red = webinterface.ledsettings.gradient_start["red"]
1152     gradient_start_green = webinterface.ledsettings.gradient_start["green"]
1153     gradient_start_blue = webinterface.ledsettings.gradient_start["blue"]
1154     gradient_start_color = wc.rgb_to_hex((int(gradient_start_red), int(gradient_start_green), int(
1155     response["gradient_start_color"] = gradient_start_color
1156

```



```

1157     gradient_end_red = webinterface.ledsettings.gradient_end["red"]
1158     gradient_end_green = webinterface.ledsettings.gradient_end["green"]
1159     gradient_end_blue = webinterface.ledsettings.gradient_end["blue"]
1160     gradient_end_color = wc.rgb_to_hex((int(gradient_end_red), int(gradient_end_green), int(gradient_end_blue)))
1161     response["gradient_end_color"] = gradient_end_color
1162
1163     key_in_scale_red = webinterface.ledsettings.key_in_scale["red"]
1164     key_in_scale_green = webinterface.ledsettings.key_in_scale["green"]
1165     key_in_scale_blue = webinterface.ledsettings.key_in_scale["blue"]
1166     key_in_scale_color = wc.rgb_to_hex((int(key_in_scale_red), int(key_in_scale_green), int(key_in_scale_blue)))
1167     response["key_in_scale_color"] = key_in_scale_color
1168
1169     key_not_in_scale_red = webinterface.ledsettings.key_not_in_scale["red"]
1170     key_not_in_scale_green = webinterface.ledsettings.key_not_in_scale["green"]
1171     key_not_in_scale_blue = webinterface.ledsettings.key_not_in_scale["blue"]
1172     key_not_in_scale_color = wc.rgb_to_hex(
1173         (int(key_not_in_scale_red), int(key_not_in_scale_green), int(key_not_in_scale_blue)))
1174     response["key_not_in_scale_color"] = key_not_in_scale_color
1175
1176     response["scale_key"] = webinterface.ledsettings.scale_key
1177
1178     response["led_color"] = led_color
1179     response["color_mode"] = color_mode
1180     response["light_mode"] = light_mode
1181     response["fading_speed"] = fading_speed
1182     response["multicolor"] = multicolor
1183     response["multicolor_range"] = multicolor_range
1184     response["rainbow_scale"] = rainbow_scale
1185     response["rainbow_offset"] = rainbow_offset
1186     response["rainbow_timeshift"] = rainbow_timeshift
1187     return jsonify(response)
1188
1189
1190 @webinterface.route('/api/get_settings', methods=['GET'])
1191 def get_settings():
1192     response = {}
1193
1194     red = webinterface.usersettings.get_setting_value("red")
1195     green = webinterface.usersettings.get_setting_value("green")
1196     blue = webinterface.usersettings.get_setting_value("blue")
1197     led_color = wc.rgb_to_hex((int(red), int(green), int(blue)))
1198
1199     backlight_red = webinterface.usersettings.get_setting_value("backlight_red")
1200     backlight_green = webinterface.usersettings.get_setting_value("backlight_green")
1201     backlight_blue = webinterface.usersettings.get_setting_value("backlight_blue")
1202     backlight_color = wc.rgb_to_hex((int(backlight_red), int(backlight_green), int(backlight_blue)))
1203
1204     sides_red = webinterface.usersettings.get_setting_value("adjacent_red")
1205     sides_green = webinterface.usersettings.get_setting_value("adjacent_green")

```

```
1206 sides_blue = webinterface.usersettings.get_setting_value("adjacent_blue")
1207 sides_color = wc.rgb_to_hex((int(sides_red), int(sides_green), int(sides_blue)))
1208
1209 light_mode = webinterface.usersettings.get_setting_value("mode")
1210 fading_speed = webinterface.usersettings.get_setting_value("fadingspeed")
1211
1212 brightness = webinterface.usersettings.get_setting_value("brightness_percent")
1213 backlight_brightness = webinterface.usersettings.get_setting_value("backlight_brightness_perce
1214
1215 response["led_color"] = led_color
1216 response["light_mode"] = light_mode
1217 response["fading_speed"] = fading_speed
1218
1219 response["brightness"] = brightness
1220 response["backlight_brightness"] = backlight_brightness
1221 response["backlight_color"] = backlight_color
1222
1223 response["sides_color_mode"] = webinterface.usersettings.get_setting_value("adjacent_mode")
1224 response["sides_color"] = sides_color
1225
1226 response["input_port"] = webinterface.usersettings.get_setting_value("input_port")
1227 response["play_port"] = webinterface.usersettings.get_setting_value("play_port")
1228
1229 response["skipped_notes"] = webinterface.usersettings.get_setting_value("skipped_notes")
1230 response["note_offsets"] = webinterface.usersettings.get_setting_value("note_offsets")
1231 response["led_count"] = webinterface.usersettings.get_setting_value("led_count")
1232 response["led_shift"] = webinterface.usersettings.get_setting_value("shift")
1233 response["led_reverse"] = webinterface.usersettings.get_setting_value("reverse")
1234
1235 response["color_mode"] = webinterface.usersettings.get_setting_value("color_mode")
1236
1237 response["multicolor"] = webinterface.usersettings.get_setting_value("multicolor")
1238 response["multicolor_range"] = webinterface.usersettings.get_setting_value("multicolor_range")
1239
1240 response["rainbow_offset"] = webinterface.usersettings.get_setting_value("rainbow_offset")
1241 response["rainbow_scale"] = webinterface.usersettings.get_setting_value("rainbow_scale")
1242 response["rainbow_timeshift"] = webinterface.usersettings.get_setting_value("rainbow_timeshift")
1243
1244 speed_slowest_red = webinterface.usersettings.get_setting_value("speed_slowest_red")
1245 speed_slowest_green = webinterface.usersettings.get_setting_value("speed_slowest_green")
1246 speed_slowest_blue = webinterface.usersettings.get_setting_value("speed_slowest_blue")
1247 speed_slowest_color = wc.rgb_to_hex((int(speed_slowest_red), int(speed_slowest_green), int(spe
1248 response["speed_slowest_color"] = speed_slowest_color
1249
1250 speed_fastest_red = webinterface.usersettings.get_setting_value("speed_fastest_red")
1251 speed_fastest_green = webinterface.usersettings.get_setting_value("speed_fastest_green")
1252 speed_fastest_blue = webinterface.usersettings.get_setting_value("speed_fastest_blue")
1253 speed_fastest_color = wc.rgb_to_hex((int(speed_fastest_red), int(speed_fastest_green), int(spe
1254 response["speed_fastest_color"] = speed_fastest_color
```

```

1255
1256     gradient_start_red = webinterface.usersettings.get_setting_value("gradient_start_red")
1257     gradient_start_green = webinterface.usersettings.get_setting_value("gradient_start_green")
1258     gradient_start_blue = webinterface.usersettings.get_setting_value("gradient_start_blue")
1259     gradient_start_color = wc.rgb_to_hex((int(gradient_start_red), int(gradient_start_green), int(
1260     response["gradient_start_color"] = gradient_start_color
1261
1262     gradient_end_red = webinterface.usersettings.get_setting_value("gradient_end_red")
1263     gradient_end_green = webinterface.usersettings.get_setting_value("gradient_end_green")
1264     gradient_end_blue = webinterface.usersettings.get_setting_value("gradient_end_blue")
1265     gradient_end_color = wc.rgb_to_hex((int(gradient_end_red), int(gradient_end_green), int(gradient_end_blue)))
1266     response["gradient_end_color"] = gradient_end_color
1267
1268     key_in_scale_red = webinterface.usersettings.get_setting_value("key_in_scale_red")
1269     key_in_scale_green = webinterface.usersettings.get_setting_value("key_in_scale_green")
1270     key_in_scale_blue = webinterface.usersettings.get_setting_value("key_in_scale_blue")
1271     key_in_scale_color = wc.rgb_to_hex((int(key_in_scale_red), int(key_in_scale_green), int(key_in_scale_blue)))
1272     response["key_in_scale_color"] = key_in_scale_color
1273
1274     key_not_in_scale_red = webinterface.usersettings.get_setting_value("key_not_in_scale_red")
1275     key_not_in_scale_green = webinterface.usersettings.get_setting_value("key_not_in_scale_green")
1276     key_not_in_scale_blue = webinterface.usersettings.get_setting_value("key_not_in_scale_blue")
1277     key_not_in_scale_color = wc.rgb_to_hex(
1278         (int(key_not_in_scale_red), int(key_not_in_scale_green), int(key_not_in_scale_blue)))
1279     response["key_not_in_scale_color"] = key_not_in_scale_color
1280
1281     response["scale_key"] = webinterface.usersettings.get_setting_value("scale_key")
1282
1283     response["speed_max_notes"] = webinterface.usersettings.get_setting_value("speed_max_notes")
1284     response["speed_period_in_seconds"] = webinterface.usersettings.get_setting_value("speed_period_in_seconds")
1285
1286     return jsonify(response)
1287
1288
1289 @webinterface.route('/api/get_recording_status', methods=['GET'])
1290 def get_recording_status():
1291     response = {}
1292     response["input_port"] = webinterface.usersettings.get_setting_value("input_port")
1293     response["play_port"] = webinterface.usersettings.get_setting_value("play_port")
1294
1295     response["isrecording"] = webinterface.saving.isrecording
1296
1297     response["isplaying"] = webinterface.saving.is_playing_midi
1298
1299     return jsonify(response)
1300
1301 @webinterface.route('/api/get_learning_status', methods=['GET'])
1302 def get_learning_status():
1303     response = {}

```

```

1304     response["loading"] = webinterface.learning.loading
1305     response["practice"] = webinterface.usersettings.get_setting_value("practice")
1306     response["hands"] = webinterface.usersettings.get_setting_value("hands")
1307     response["mute_hand"] = webinterface.usersettings.get_setting_value("mute_hand")
1308     response["start_point"] = webinterface.usersettings.get_setting_value("start_point")
1309     response["end_point"] = webinterface.usersettings.get_setting_value("end_point")
1310     response["set_tempo"] = webinterface.usersettings.get_setting_value("set_tempo")
1311     response["hand_colorR"] = webinterface.usersettings.get_setting_value("hand_colorR")
1312     response["hand_colorL"] = webinterface.usersettings.get_setting_value("hand_colorL")
1313     response["hand_colorList"] = ast.literal_eval(webinterface.usersettings.get_setting_value("hand_colorList"))
1314     response["is_loop_active"] = ast.literal_eval(webinterface.usersettings.get_setting_value("is_loop_active"))
1315
1316     return jsonify(response)
1317
1318
1319 @webinterface.route('/api/get_songs', methods=['GET'])
1320 def get_songs():
1321     page = request.args.get('page')
1322     page = int(page) - 1
1323     length = request.args.get('length')
1324     sortby = request.args.get('sortby')
1325     search = request.args.get('search')
1326
1327     start = int(page) * int(length)
1328
1329     songs_list_dict = {}
1330
1331     path = 'Songs/'
1332     songs_list = os.listdir(path)
1333     songs_list = [os.path.join(path, i) for i in songs_list]
1334
1335     songs_list = sorted(songs_list, key=os.path.getmtime)
1336
1337     if sortby == "dateAsc":
1338         songs_list.reverse()
1339
1340     if sortby == "nameAsc":
1341         songs_list.sort()
1342
1343     if sortby == "nameDesc":
1344         songs_list.sort(reverse=True)
1345
1346     i = 0
1347     total_songs = 0
1348
1349     for song in songs_list:
1350         if "_" in song or not song.endswith('.mid'):
1351             continue
1352         if search:

```

```

1353         if search.lower() not in song.lower():
1354             continue
1355         total_songs += 1
1356
1357     max_page = int(math.ceil(total_songs / int(length)))
1358
1359     for song in songs_list:
1360         song = song.replace("Songs/", "")
1361         date = os.path.getmtime("Songs/" + song)
1362         if "_" in song or not song.endswith('.mid'):
1363             continue
1364
1365         if search:
1366             if search.lower() not in song.lower():
1367                 continue
1368
1369         i += 1
1370         if (i > int(start)):
1371             songs_list_dict[song] = date
1372
1373         if len(songs_list_dict) >= int(length):
1374             break
1375
1376     return render_template('songs_list.html', len=len(songs_list_dict), songs_list_dict=songs_list_dict,
1377                           max_page=max_page, total_songs=total_songs)
1378
1379
1380 @webinterface.route('/api/get_ports', methods=['GET'])
1381 def get_ports():
1382     ports = mido.get_input_names()
1383     ports = list(dict.fromkeys(ports))
1384     response = {}
1385     response["ports_list"] = ports
1386     response["input_port"] = webinterface.usersettings.get_setting_value("input_port")
1387     response["secondary_input_port"] = webinterface.usersettings.get_setting_value("secondary_input_port")
1388     response["play_port"] = webinterface.usersettings.get_setting_value("play_port")
1389     response["connected_ports"] = str(subprocess.check_output(["aconnect", "-i", "-l"]))
1390
1391     return jsonify(response)
1392
1393
1394 @webinterface.route('/api/switch_ports', methods=['GET'])
1395 def switch_ports():
1396     active_input = webinterface.usersettings.get_setting_value("input_port")
1397     secondary_input = webinterface.usersettings.get_setting_value("secondary_input_port")
1398     webinterface.midiports.change_port("input", secondary_input)
1399     webinterface.usersettings.change_setting_value("secondary_input_port", active_input)
1400     webinterface.usersettings.change_setting_value("input_port", secondary_input)
1401

```

```

1402     fastColorWipe(webinterface.ledstrip.strip, True, webinterface.ledsettings)
1403
1404     return jsonify(success=True)
1405
1406
1407 @webinterface.route('/api/get_sequences', methods=['GET'])
1408 def get_sequences():
1409     response = {}
1410     sequences_list = []
1411     sequences_tree = minidom.parse("sequences.xml")
1412     i = 0
1413     while True:
1414         try:
1415             i += 1
1416             sequences_list.append(
1417                 sequences_tree.getElementsByTagName("sequence_" + str(i))[0].getElementsByTagName(
1418                     "sequence_name")[
1419                     0].firstChild.nodeValue)
1420         except:
1421             break
1422     response["sequences_list"] = sequences_list
1423     response["sequence_number"] = webinterface.ledsettings.sequence_number
1424
1425     return jsonify(response)
1426
1427
1428 @webinterface.route('/api/get_steps_list', methods=['GET'])
1429 def get_steps_list():
1430     response = {}
1431     sequence = request.args.get('sequence')
1432     sequences_tree = minidom.parse("sequences.xml")
1433     steps_list = []
1434     i = 0
1435
1436     for step in sequences_tree.getElementsByTagName("sequence_" + str(sequence))[0].childNodes:
1437         if (step.nodeType == 1):
1438             if (step.nodeName == "settings"):
1439                 response["control_number"] = step.getElementsByTagName("control_number")[0].firstC
1440                 response["next_step"] = step.getElementsByTagName("next_step")[0].firstChild.nodeV
1441             else:
1442                 steps_list.append(step.nodeName)
1443
1444     response["steps_list"] = steps_list
1445     return jsonify(response)
1446
1447
1448 @webinterface.route('/api/set_step_properties', methods=['GET'])
1449 def set_step_properties():
1450     sequence = request.args.get('sequence')

```

```
1451     step = request.args.get('step')
1452     webinterface.ledsettings.set_sequence(sequence, step, True)
1453
1454     return jsonify(success=True)
1455
1456
1457 def pretty_print(dom):
1458     return '\n'.join([line for line in dom.toprettyxml(indent=' ' * 4).split('\n') if line.strip()])
1459
1460
1461 def pretty_save(file_path, sequences_tree):
1462     with open(file_path, "w", encoding="utf8") as outfile:
1463         outfile.write(pretty_print(sequences_tree))
```