

```
1 """Contient le cœur de l'application
2
3 Toutes les routes de l'application sont décrites dans ce
4 fichier.
5
6 @author: Tanguy Cavagna
7 @copyright: Copyright 2020, TPI
8 @version: 1.0.0
9 @date: 2020-05-26
10 """
11 from flask import Flask, url_for, redirect, flash, g
12 from flask_login import LoginManager
13 from dotenv import load_dotenv
14
15 from .config import DevConfig, ProdConfig
16 from .packages.controllers.UserController import
17 UserController
18
19 app = Flask(__name__)
20 app.config.from_object(DevConfig())
21 app.secret_key =
22     8185c8ac4656219f4aa5541915079f7b3743e1b5f48bacfcc3386af016b553
23     20"
24
25 login_manager = LoginManager()
26 login_manager.init_app(app)
27
28 # Chargement des routes
29 with app.app_context():
30     from . import routes, auth
31
32     # Enregistre les routes
33     app.register_blueprint(routes.main_bp)
34     app.register_blueprint(auth.auth_bp)
35
36     # =====
37     # OTHER
38     # =====
39     @login_manager.user_loader
40     def load_user(user_id):
41         """Test si l'utilisateur est connecté sur toutes les pages
42         . Si oui, retourne l'utilisateur"""
43         if user_id is not None:
44             return UserController().get_by_id(user_id)
45         return None
46
47     # =====
```

```
43 #         Run
44 # =====
45 if __name__ == "__main__":
46     app.run()
```

```
1 """Contient les routes d'authentification de l'application
2
3 @author: Tanguy Cavagna
4 @copyright: Copyright 2020, TPI
5 @version: 1.0.0
6 @date: 2020-05-27
7 """
8 import hashlib
9 import re
10 from typing import Any
11
12 from flask import redirect, render_template, flash, Blueprint,
13 , request, url_for
13 from flask_login import login_required, current_user,
14 login_user, logout_user
14
15 from .packages.controllers.UserController import
16 UserController
16 from .packages.controllers.authentication import
17 SignupController, LoginController
17
18 # Configuration du Blueprint
19 auth_bp = Blueprint('auth_bp', __name__,
20                     template_folder='templates',
21                     static_folder='static')
22
23 @auth_bp.route('/signup', methods=['GET', 'POST'])
24 def signup():
25     """Affiche la page d'inscription
26     \nGET: Affiche la page
27     \nPOST: Si les informations de connexion sont valides
28     , redirection sur l'accueil
28 """
29     auth_validation = SignupController(request.form)
30
31     if request.method == 'POST':
32         if auth_validation.validate():
33             email = request.form.get('email')
34             nickname = request.form.get('nickname')
35             password = request.form.get('password')
36
37             if not UserController().exists(email):
38                 hashed_password = hashlib.sha256(password.
39 encode('utf8')).hexdigest()
40                 user = UserController().insert(email, nickname
40 , hashed_password)
40                 UserController().setup_default_lists(user.id)
```

```

41
42             if user is not None:
43                 login_user(user)
44                 return redirect(url_for('main_bp.home'))
45
46             flash('Un utilisateur utilise déjà cet email')
47             return redirect(url_for('auth_bp.signup'))
48
49     return render_template('signup.html',
50                           current_user=current_user,
51                           auth_errors=auth_validation.
52                           get_state())
53
54 @auth_bp.route('/login', methods=['GET', 'POST'])
55 def login():
56     """Affiche la page de connexion
57         \nGET: Affiche la page
58         \nPOST: Si les informations de connexion sont valides
59         , redirection sur l'accueil
60         """
61
62     auth_validation = LoginController(request.form)
63
64     if request.method == 'POST':
65         if auth_validation.validate():
66             email = request.form.get('email')
67             password = request.form.get('password')
68             hashed_password = hashlib.sha256(password.encode(
69                 'utf8')).hexdigest()
70
71             if UserController().check_password(email,
72                 hashed_password):
73                 user = UserController().get_by_email(email)
74                 login_user(user)
75                 return redirect(url_for('main_bp.home'))
76
77             flash('Combinaison email - mot de passe invalide')
78             return redirect(url_for('auth_bp.login'))
79
80     return render_template('login.html',
81                           current_user=current_user,
82                           auth_errors=auth_validation.
83                           get_state())
84
85 @auth_bp.route('/logout', methods=['GET'])
86 @login_required
87 def logout():
88     """Déconnecte l'utilisateur"""

```

```
83     logout_user()
84     return redirect(url_for('auth_bp.login'))
85
```

```
1 """Contient les différentes configuration du serveur Flask
2
3 @author: Tanguy Cavagna
4 @copyright: Copyright 2020, TPI
5 @version: 1.0.0
6 @date: 2020-05-26
7 """
8 class ProdConfig:
9     TESTING = False
10    DEBUG = False
11    TEMPLATES_AUTO_RELOAD = False
12
13 class DevConfig:
14    TESTING = True
15    DEBUG = True
16    TEMPLATES_AUTO_RELOAD = True
```

```

1 """Contient les routes de l'application
2
3 @author: Tanguy Cavagna
4 @copyright: Copyright 2020, TPI
5 @version: 1.0.0
6 @date: 2020-05-26
7 """
8 from flask import render_template, Response,
9     stream_with_context, jsonify, request, Blueprint, redirect,
10    url_for
11 from flask import current_app as app
12 from flask_swagger import swagger
13 from flask_login import current_user, login_required
14 import markdown, os
15
16 from .packages.controllers.AnimeController import
17     AnimeController
18 from .packages.controllers.ListController import
19     ListController
20 from .packages.controllers.UserController import
21     UserController
22 from .packages.controllers.ActivitiesController import
23     ActivitiesController
24 from .packages.controllers.SqliteController import
25     SqliteController
26
27 # Configuration du Blueprint
28 main_bp = Blueprint('main_bp', __name__,
29                      template_folder='templates',
30                      static_folder='static')
31
32 # =====
33 #      STREAM
34 # =====
35
36 def stream_template(template_name, **context):
37     """Le but est de récupérer l'objet template de Jinja2 et d
38     'appeler la fonction stream() qui renvoie un stream au lieu de
39     render() qui renvoie un string.
40     Étant donné que l'on bypass le template de Flask, on doit
41     être sûre de mettre à jour de template nous même en appelant
42     update_template_context().
43     Le template sera évalué chaque fois que le stream sera
44     modifier. À chaque yield, le serveur va envoyer le contenu au
45     client et donc modifier le stream.
46
47     Arguments:
48         template_name {string} -- Nom du template a appeler

```

```
35          **context -- Tous les autres argument mixes  
36          représentant les variables à envoyer au template  
37      Returns:  
38          stream -- Stream à envoyer  
39          """  
40      # https://flask.palletsprojects.com/en/1.1.x/patterns/  
41      # streaming/  
42      app.update_template_context(context)  
43      t = app.jinja_env.get_template(template_name)  
44      rv = t.stream(context)  
45      return rv  
46  # =====  
47  #       GET  
48  # =====  
49 @main_bp.route('/')
```

50 **def** home():
51 """Affiche la page d'accueil
52 """
53 return render_template('index.html',
54 current_user=current_user)
55

56 @main_bp.route('/search/<string:search_string>', methods=['GET'])

57 @login_required

58 **def** search(search_string: str = **None**):
59 """Recherche un anime
60 """
61 # Parse la chaine de recherche pour transformer la chaine
62 # comme suit (facilite la recherche fulltext) :
63 # Chaine de base : Ma chaine de recherche
64 # Chaine parsée : "Ma" "chaine" "de" "recherche"
65 search_string_parsed = ' '.join(['' + escaped + '' for
66 escaped **in** search_string.split(' ')])
67 searched_animes = AnimeController().get_from_search_string
68 (search_string_parsed)

69 # Ajout des relations
70 **for** anime **in** searched_animes:
71 anime['relations'] = AnimeController().
72 get_relations_by_anime_id(anime['id'])
73 anime['is_favorite'] = AnimeController().
74 is_anime_in_user_favorite(current_user.id, anime['id'])
75 anime['in_list'] = [l['id'] **for** l **in** ListController().
76 get_list_of_an_anime(anime['id'])]

```

73     return render_template('index.html',
74                             current_user=current_user,
75                             search_string=search_string,
76                             search_results=searched_animes,
77                             user_list=ListController().
78                             get_user_lists(current_user.id))
78
79 @main_bp.route('/random', methods=['GET'])
80 @login_required
81 def random():
82     """Affiche un anime random
83     """
84     random_anime = AnimeController().get_random_anime()
85
86     random_anime['relations'] = AnimeController().
87     get_relations_by_anime_id(random_anime['id'])
87     random_anime['is_favorite'] = AnimeController().
88     is_anime_in_user_favorite(current_user.id, random_anime['id'])
88     random_anime['in_list'] = [l['id'] for l in
89     ListController().get_list_of_an_anime(random_anime['id'])]
89
90     return render_template('index.html',
91                             current_user=current_user,
92                             search_string='aléatoire',
93                             search_results=[random_anime],
94                             user_list=ListController().
95                             get_user_lists(current_user.id))
95
96 @main_bp.route('/profile/<string:nickname>', methods=['GET'])
97 @login_required
98 def profile(nickname: str = None):
99     """Affiche la page de profile de l'utilisateur mentionner
100    dans l'url
100
101    if nickname is None:
102        nickname = current_user.nickname
103
104    searched_user = UserController().get_by_nickname(nickname
104    )
105
106    if searched_user is not None:
107        return render_template('profile.html',
108                                searched_user=searched_user,
109                                stats=UserController().
110                                get_stats_by_id(searched_user.id))
110

```

```

111     return redirect(url_for('main_bp.home'))
112
113 @main_bp.route('/profile', methods=['GET'])
114 @main_bp.route('/profile/', methods=['GET'])
115 @login_required
116 def profile_redirect():
117     """Redirige l'utilisateur sur sa page de profile si aucun
118     pseudo indiqué dans l'url
119     """
120
121     return redirect(url_for('main_bp.profile', nickname=
122     current_user.nickname))
123
124 @main_bp.route('/lists/<string:nickname>')
125 @login_required
126 def lists(nickname: str = None):
127     """Affiche toutes les liste et leur contenu de l'
128     utilisateur mentionné dans l'url
129     """
130
131     if nickname is None:
132         nickname = current_user.nickname
133
134     searched_user = UserController().get_by_nickname(nickname)
135
136     if searched_user is not None:
137         user_lists = ListController().get_user_lists(
138             searched_user.id)
139
140         return render_template('lists.html',
141                               current_user=current_user,
142                               user_lists=user_lists,
143                               searched_user=searched_user)
144
145     return redirect(url_for('main_bp.home'))
146
147 @main_bp.route('/lists')
148 @main_bp.route('/lists/')
149 @login_required
150 def lists_redirect():
151     """Redirige l'utilisateur si aucun pseudo n'est inscrit
152     dans l'url
153     """
154
155     return redirect(url_for('main_bp.lists', nickname=
156     current_user.nickname))
157
158 @main_bp.route('/favorites/<string:nickname>')
159 @login_required
160 def favorites(nickname: str = None):

```

```

151     """Affiche tout les favoris de l'utilisateur mentionné
152     dans l'url
153     """
154     if nickname is None:
155         nickname = current_user.nickname
156
157     searched_user = UserController().get_by_nickname(nickname)
158
159     if searched_user is not None:
160         user_lists = ListController().get_user_lists(
161             searched_user.id)
162         return render_template('favorites.html',
163                               current_user=current_user,
164                               searched_user=searched_user)
165
166     @main_bp.route('/favorites')
167     @main_bp.route('/favorites/')
168     @login_required
169     def favorites_redirect():
170         """Redirige l'utilisateur si aucun pseudo n'est inscrit
171         dans l'url
172         """
173
174     @main_bp.route('/about')
175     def about():
176         """Affiche la page à propos
177         """
178         with open(os.path.dirname(os.path.realpath(__file__)) +
179                   '/docs/librairies.md', 'r', encoding='utf-8') as input_file:
180             text = input_file.read()
181
182         return render_template('about.html', current_user=
183                               current_user, html=markdown.markdown(text))
184
185     @main_bp.route('/help')
186     def help():
187         """Affiche la page d'aide
188         """
189         with open(os.path.dirname(os.path.realpath(__file__)) +
190                   '/docs/aide.md', 'r', encoding='utf-8') as input_file:
191             text = input_file.read()

```

```

190     return render_template('help.html', current_user=
191         current_user, html=markdown.markdown(text))
192 @main_bp.route('/get/favorites', methods=['GET'])
193 @login_required
194 def getFavorites_for_loged_user():
195     """
196     Récupère tous les favoris de l'utilisateur connecté
197     ---
198     tags:
199         - favorites
200     responses:
201         200:
202             $ref: '/static/swagger-components/components.yml#/components/responses/LoggedUserFavorites'
203             """
204     return jsonify({'animes': AnimeController().
205         get_favorite_by_user_id(current_user.id)})
206 @main_bp.route('/get/favorites/<string:nickname>', methods=['GET'])
207 @login_required
208 def getFavorites_for_user(nickname = None):
209     """
210     Récupère tous les favoris d'un utilisateur
211     ---
212     tags:
213         - favorites
214     responses:
215         200:
216             $ref: '/static/swagger-components/components.yml#/components/responses/LambdaUserFavorites'
217             """
218     searched_user = UserController().get_by_nickname(nickname)
219     if searched_user is not None:
220         user_id = searched_user.id
221         return jsonify({'animes': AnimeController().
222             get_favorite_by_user_id(user_id)})
223     return jsonify({'animes': []})
224
225 @main_bp.route('/endpoints', methods=['GET'])
226 def endpoints():
227     """Affiche les points de sorties de l'api. Comprend
228     uniquement les urls documentées avec de la docstring swagger
229     """

```

```

229     return render_template('endpoints.html')
230
231 @main_bp.route("/specs", methods=['GET'])
232 def spec():
233     """
234     Retourne un JSON comportant toutes les informations
235     contenues dans les docstring des fonctions
236     ---
237     tags:
238         - API
239     response:
240         200:
241             description: objet JSON comportant toutes les
242             informations nécessaire à l'affichage des points de sortie de
243             l'API
244             """
245     swag = swagger(app)
246     swag['info'][‘version’] = "1.0"
247     swag['info'][‘title’] = "Animanga"
248     return jsonify(swag)
249
250 # J'ai été obligé de mettre la route en GET, car ce n'est
251 # pas possible de streamer des infos si elles viennent d'un
252 # post
253 # Lors de l'affichage des specs - '/enpoints' - j'ai
254 # remplacer le type de méthode de GET à PUT pour une meilleure
255 # compréhension de l'api
256 # La fonction stream_with_context() permet de garder le
257 # contexte de la requête qui se fait supprimer en tant normale
258 # . Indispensable lors de stream avec template contenant un
259 # layout
260 @main_bp.route('/override')
261 def override_datas():
262     """
263     Écrase les données existantes de la bd avec de nouvelles
264     données
265     ---
266     tags:
267         - animes
268     responses:
269         200:
270             description: Remplacement effectué et redirection sur
271             l'index
272             """
273     current_directory = '/'.join(__file__.split('/')[-1:][:-1])
274     return Response(stream_with_context(stream_template(
275         'index.html',

```

```

263     override_messages=AnimeController().override_local_with_json(
264         current_directory + '/static/json/anime.json'))))
265 @main_bp.route('/sync')
266 @login_required
267 def sync():
268     """Synchronise les données entre Sqlite3 et MySQL
269     """
270     SqliteController().synchronise_with_mysql()
271
272     return redirect(url_for('main_bp.home'))
273
274 # =====
275 #      POST, PATCH, PUT, DELETE
276 # =====
277 @main_bp.route('/set/favorite', methods=['PATCH'])
278 @login_required
279 def set_favorite():
280     """
281     Met à jour le status de favoris pour un anime de l'
282     utilisateur connecté
283     ---
284     tags:
285         - favorites
286     parameters:
287         - $ref: '/static/swagger-components/components.yml#/components/parameters/idAnime'
288     responses:
289         200:
290             $ref: '/static/swagger-components/components.yml#/components/responses/GenericResponse'
291             """
292             anime_id = request.get_json()['idAnime']
293
294             if anime_id is not None:
295                 return jsonify({'Status': AnimeController().
296                     set_anime_in_user_favorite(current_user.id, anime_id)})
297
298             return jsonify({'Status': False})
299
300 @main_bp.route('/set/list', methods=['PUT'])
301 @login_required
302 def set_list():
303     """
304     Met à jour le status de favoris d'un anime pour l'
305     utilisateur connecté

```

```

303     ---
304     tags:
305         - lists
306     parameters:
307         - $ref: '/static/swagger-components/components.yml#/components/parameters/idAnime'
308         - $ref: '/static/swagger-components/components.yml#/components/parameters/idList'
309     responses:
310         200:
311             $ref: '/static/swagger-components/components.yml#/components/responses/GenericResponse'
312             """
313             id_anime = request.get_json()['idAnime']
314             id_list = request.get_json()['idList']
315
316             if id_anime is not None:
317                 return jsonify({'Status': AnimeController().set_anime_in_list(id_anime, id_list)})
318
319             return jsonify({'Status': False})
320
321 @main_bp.route('/delete/defaults', methods=['DELETE'])
322 @login_required
323 def delete_status():
324     """
325         Supprime l'anime des listes par défaut de l'utilisateur connecté
326     ---
327     tags:
328         - animes
329     parameters:
330         - $ref: '/static/swagger-components/components.yml#/components/parameters/idAnime'
331     responses:
332         200:
333             $ref: '/static/swagger-components/components.yml#/components/responses/GenericResponse'
334             """
335             anime_id = request.get_json()['idAnime']
336
337             if anime_id is not None:
338                 return jsonify({'Status': AnimeController().delete_anime_status(current_user.id, anime_id)})
339
340             return jsonify({'Status': False})
341

```

```

342 @main_bp.route('/get/animes', methods=['POST'])
343 @login_required
344 def get_animes_from_list():
345     """
346     Récupère les animes d'une liste
347     ---
348     tags:
349         - animes
350     parameters:
351         - $ref: '/static/swagger-components/components.yml#/components/parameters/userNickname'
352             - $ref: '/static/swagger-components/components.yml#/components/parameters/idList'
353                 - $ref: '/static/swagger-components/components.yml#/components/parameters/search_terms'
354     responses:
355         200:
356             $ref: '/static/swagger-components/components.yml#/components/responses/AnimeListResponse'
357             """
358     if request.json.get('nicknameUser') is not None and
359         request.json.get('nicknameUser') != current_user.nickname:
360         user_id = UserController().get_by_nickname(request.
361             json.get('nicknameUser')).id
362     else:
363         user_id = current_user.id
364     return jsonify({ 'animes': AnimeController().
365         get_animes_in_list_for_user(user_id, request.json.get('idList')
366         ), request.json.get('search-terms') })
367 @main_bp.route('/add/list', methods=['PUT'])
368 @login_required
369 def add_list():
370     """
371     Ajoute une nouvelle liste à l'utilisateur connecté
372     ---
373     tags:
374         - lists
375     parameters:
376         - $ref: '/static/swagger-components/components.yml#/components/parameters/newListName'
377     responses:
378         200:
379             $ref: '/static/swagger-components/components.yml#/components/responses/ListResponse'
380             """

```

```

379     return jsonify({ 'list': ListController().add_new_list(
    request.json.get('newListName'), current_user.id) })
380
381 @main_bp.route('/delete/list', methods=['DELETE'])
382 @login_required
383 def delete_list():
384     """
385     Supprime une liste à l'utilisateur connecté
386     ---
387     tags:
388         - lists
389     parameters:
390         - $ref: '/static/swagger-components/components.yml#/components/parameters/idList'
391     responses:
392         200:
393             $ref: '/static/swagger-components/components.yml#/components/responses/GenericResponse'
394         """
395     return jsonify({ 'Status': ListController().delete_by_id(
    request.json.get('idList'), current_user.id) })
396
397 @main_bp.route('/set/list/name', methods=['PATCH'])
398 @login_required
399 def rename_list():
400     """Renomme une liste
401     """
402     list_id = request.get_json()['idList']
403     new_list_name = request.get_json()['newListName']
404
405     return jsonify({'Status': ListController().rename(list_id,
    new_list_name)})
406
407 @main_bp.route('/set/favorites-order', methods=['PATCH'])
408 @login_required
409 def set_favorites_order():
410     """
411     Met à jour l'ordre des favoris
412     ---
413     tags:
414         - favorites
415     parameters:
416         - $ref: '/static/swagger-components/components.yml#/components/parameters/idAnimeList'
417     responses:
418         200:
419             $ref: '/static/swagger-components/components.yml#/'

```

```
419 components/responses/GenericResponse'
420     """
421     return jsonify({ 'Status': AnimeController().
422         reorderFavorites(current_user.id, request.json.get('ids'
423     )) })
424
425 @main_bp.route('/get/activities', methods=[ 'GET' ])
426 @login_required
427 def get_activities_for_logged_user():
428     """
429     Récupère tous les activités de l'utilisateur connecté
430     ---
431     tags:
432         - activities
433     responses:
434         200:
435             $ref: '/static/swagger-components/components.yml#/components/responses/LoggedUserActivities'
436             """
437     return jsonify({ 'activities': ActivitiesController().
438         getLast24h(current_user.id) })
```

```
1 from subprocess import Popen, PIPE
2
3 # Execute la commande : python3 -m md_toc github ../compiled/
4 # Cette commande sert à générer une table des matières pour un
5 # document markdown.
6 p = Popen(['python3', '-m', 'md_toc', 'github', '../compiled/',
7 final.md'], stdin=PIPE, stdout=PIPE, stderr=PIPE)
8 # Récupère la sortie de la commande (la table des matière)
9 (output, err) = p.communicate()
10 toc = output.decode('utf-8') # Decode la sortie
11
12 # Ajoute le titre "Table des matières" juste avant la table
13 # des matières
14 toc = '## Table des matière\n\n' + toc + "\n<div style='page-
15 #break-after: always; break-after: page; text-align:right;'></
16 div>"
17
18 updated_lines = []
19 # Ajoute la table des matière dans les lignes du fichier
20 with open('../compiled/final.md', 'r+') as f:
21     lines = f.readlines()
22     del lines[0]
23     del lines[1]
24     f.seek(0)
25     lines.insert(0, toc)
26     updated_lines = lines
27     f.truncate(0) # Suppression de l'ancien contenu
28     f.writelines(updated_lines) # Ajout du nouveau contenu
29     f.close()
```

```
1 #!/bin/bash
2
3 # Fichier final
4 final=../compiled/final.md
5
6 # Écrase le fichier
7 echo "" > $final
8
9 # Parcour tout les fichiers md souhaité dans un certain ordre
10 for f in ../index.md \
11     ../resume-enonce.md \
12     ../methodologie.md \
13     ../planning.md \
14     ../implementation.md \
15     ../librairies.md \
16     ../fonctionnalites-majeurs.md \
17     ../scenarios-tests.md \
18     ../bibliographie.md \
19     ../conclusion.md
20 do
21     echo >> $final
22     cat $f >> $final
23 done
24
25 # Exécuter le script python pour générer la table des matières
26 exec python3 toc.py
27
```

```
1 #!/bin/sh
2
3 # Fusionne la page de garde avec le contenu du document final
4 pdftk ../compiled/cover.pdf \
5     ../compiled/final.pdf \
6     ../compiled/annexes-cover.pdf \
7     ../compiled/resume-cover.pdf \
8     ../compiled/resume-tpi.pdf \
9     ../compiled/enonce-cover.pdf \
10    ../compiled/"Enoncé TPI CAVAGNA Tanguy I.DA-P4B v2.
11 pdf" \
12    ../compiled/journal-de-bord-cover.pdf \
13    ../compiled/journal-de-bord.pdf \
14    ../compiled/code-source-cover.pdf \
15    ../compiled/code-source.pdf \
16    cat output \
17    ../compiled/rapport-tpi-et-documentation-technique.
pdf
```

```
1 /**
2  * Script pour la prise en charge des raccourcis clavier
3 *
4  * @author: Tanguy Cavagna
5  * @copyright: Copyright 2020, TPI
6  * @version: 1.0.0
7  * @date: 2020-05-28
8 */
9
10 /**
11 * Prise en charge des évènements keydown
12 *
13 * @param {event} e Évènement keyup
14 */
15 function keydownHandler(e) {
16     // Prise en charge du CTRL+S
17     if (e.ctrlKey && e.keyCode === 83) {
18         e.preventDefault(); // Annule l'event de sauvegarde
19         natif
20         const searchModalTrigger = document.getElementById('
21             search-modal-trigger');
22         if (searchModalTrigger !== undefined) {
23             searchModalTrigger.click();
24         }
25     }
26 }
27
28 document.addEventListener('keydown', keydownHandler, false);
29
```

```
1 /**
2  * Script permettant le redimensionnement des images en
3  * fonction de la taille de l'écran
4  *
5  * @author: Tanguy Cavagna
6  * @copyright: Copyright 2020, TPI
7  * @version: 1.0.0
8  * @date: 2020-05-28
9 */
10 const resultsItemPictures = document.querySelectorAll('.search-
11 results .result-item .left .picture');
12
13 /**
14  * Redéfini la taille des images pour garder un ratio correct
15 */
16 function resizeThumbnails() {
17     let newHeight = -1;
18     // Re defini la hauteur de l'image en fonction de sa
19     // largeur
20     resultsItemPictures.forEach((item) => {
21         const currentItem = item;
22         newHeight = parseInt(currentItem.getBoundingClientRect().width / 0.7333, 10);
23         currentItem.style.height = `${newHeight}px`;
24         // Defini la hauteur max de la carte en fonction de la
25         // hauteur de l'image principale
26         resultsItem.forEach((item) => {
27             const currentItem = item;
28             currentItem.style.maxHeight = `${currentItem.
29             querySelector('.left .picture').getBoundingClientRect().height
30             }px`;
31         });
32     });
33
34 window.addEventListener('load', resizeThumbnails);
35 window.addEventListener('resize', resizeThumbnails);
36
```

```

1 /* global fetchFavorites */
2
3 /**
4  * Script pour la prise en charge du chargement des favoris,
5  * ajout de favoris,
6  * ajout d'anime dans les listes
7  *
8  * @author: Tanguy Cavagna
9  * @copyright: Copyright 2020, TPI
10 * @version: 1.0.0
11 * @date: 2020-05-29
12 */
13 const favoriteTogglers = document.querySelectorAll('.favorite-toggler');
14 const statusCombo = document.querySelectorAll('.status_combo');
15 const customChecks = document.querySelectorAll('.custom-check');
16
17 /**
18  * Supprime un anime des listes par défaut
19  *
20  * @param {int} idAnime Id de l'anime à supprimer des listes
21  * par défaut
22  * @param {function} callback Function à exécuter en tant que
23  * callback du fetch
24  */
25 function deleteAnimeFromDefaultLists(idAnime, callback
26   = () => {}) {
27   return fetch('/delete/defaults', {
28     method: 'DELETE',
29     headers: {
30       Accept: 'application/json',
31       'Content-Type': 'application/json',
32     },
33     body: JSON.stringify({ idAnime }),
34   }).then((response) => response.json()).then(() => {
35     callback();
36   });
37 /**
38  * Ajout un anime dans une liste
39  *
40  * @param {int} idAnime Id de l'anime
41  * @param {int} idList Id de la liste

```

```

41 * @param {function} callback Fonction à exécuter en tant que
42 * callback du fetch
43 function putAnimeInList(idAnime, idList, callback
44   = () => {}) {
45   fetch('/set/list', {
46     method: 'PUT',
47     headers: {
48       Accept: 'application/json',
49       'Content-Type': 'application/json',
50     },
51     body: JSON.stringify({ idAnime, idList }),
52   }).then((response) => response.json()).then((json) => {
53     callback(json);
54   });
55
56 // Changement du status de favoris de l'anime
57 favoriteTogglers.forEach((favoriteToggler) => {
58   favoriteToggler.addEventListener('click', (e) => {
59     const toggler = e.target.closest('.favorite-toggler');
60
61     fetch('/set/favorite', {
62       method: 'PATCH',
63       headers: {
64         Accept: 'application/json',
65         'Content-Type': 'application/json',
66       },
67       body: JSON.stringify({ idAnime: toggler.dataset.id
68     }),           });
69     }).then((response) => response.json()).then((json
70 ) => {
71   // Check le toggle favoris selon l'état de l'anime
72   if (json.Status === true) {
73     toggler.classList.add('is-favorite');
74   } else {
75     toggler.classList.remove('is-favorite');
76   }
77   fetchFavorites();
78 });
79 });
80
81 // Changement du statut de visionnement de l'anime. (Complété
82 , En cours, Abondoné, Planifié)
83 statusCombo.forEach((comboOption) => {

```

```
83     comboOption.addEventListener('change', async () => {
84         const idAnime = comboOption.value.split('-')[0];
85         const idList = comboOption.value.split('-')[1];
86
87
88         if (idList !== 'none') {
89             await deleteAnimeFromDefaultLists(idAnime);
90             putAnimeInList(idAnime, idList);
91         } else {
92             deleteAnimeFromDefaultLists(idAnime);
93         }
94     });
95 });
96
97 // Ajout d'un anime dans une liste
98 customChecks.forEach((customCheck) => {
99     customCheck.addEventListener('click', (e) => {
100         const check = e.target;
101         const input = check.querySelector('input[type="checkbox"]');
102         input.checked = !input.checked;
103
104         const idAnime = input.value.split('-')[0];
105         const idList = input.value.split('-')[1];
106
107         putAnimeInList(idAnime, idList, (json) => {
108             if (json.Status === true) {
109                 check.classList.add('checked');
110             } else {
111                 check.classList.remove('checked');
112             }
113         });
114     });
115 });
116
```

```
1 /**
2  * Script générant une table des matières
3  *
4  * @author: Tanguy Cavagna
5  * @copyright: Copyright 2020, TPI
6  * @version: 1.0.0
7  * @date: 2020-06-08
8 */
9 const titles = document.querySelectorAll('h1, h2, h3, h4, h5,
10 const toc = document.querySelector('.toc');
11
12 for (let i = 0; i < titles.length; i += 1) {
13     const title = titles[i];
14     const tocTitle = document.createElement(title.tagName.
    toLowerCase());
15     const tocLink = document.createElement('a');
16
17     if (!title.hasAttribute('id')) {
18         title.id = `h${i}`;
19     }
20     tocLink.href = `#${title.getAttribute('id')}`;
21     tocLink.innerHTML = title.textContent;
22     tocTitle.appendChild(tocLink);
23     toc.appendChild(tocTitle);
24 }
25
```

```
1 /**
2  * Script pour la prise en charge de la recherche d'anime
3  *
4  * @author: Tanguy Cavagna
5  * @copyright: Copyright 2020, TPI
6  * @version: 1.0.0
7  * @date: 2020-05-29
8 */
9
10 const searchClear = document.querySelector('.search-clear');
11 const searchString = document.querySelector('.search-form .
  search-string');
12 const pickRandom = document.querySelector('#pick-random');
13
14 if (searchClear != null) {
15     searchClear.addEventListener('click', () => {
16         searchString.value = '';
17     });
18 }
19
20 if (pickRandom != null) {
21     pickRandom.addEventListener('click', () => {
22         window.location.replace('/random');
23     });
24 }
25
26 if (searchString != null) {
27     // Redirige sur la page de recherche lors de l'appui sur
28     // la touche entrée
29     // dans le champs de recherche
30     searchString.onkeydown = (e) => {
31         if (e.keyCode === 13) { // Touche entrée
32             window.location.replace(`/search/${searchString.
  value}`);
33         }
34     };
35 }
```

```

1  /**
2   * Récupère les animes favoris de l'utilisateur
3   *
4   * @param {boolean} col Est-ce l'affichage avec colonne de
5   * Bootstrap doit être utilisé
6   * @param {string} searchedUser Pseudo de l'utilisateur
7   * cherché si sur la page d'un autre
8   * utilisateur que celui connecté
9   */
10  /* eslint-disable-next-line no-unused-vars */
11  function fetchFavorites(col = false, canDelete = false,
12    searchedUser = null) {
13    const favorites = document.querySelector('.favorites');
14
15    if (favorites != null) {
16      let url = '/get/favorites';
17
18      // Ajout du slug `nickname` si l'on veut récupérer les
19      // favoris d'un autre utilisateur que celui connecté
20      if (searchedUser != null) {
21        url += `/ ${searchedUser}`;
22      }
23
24      fetch(url, {
25        method: 'GET',
26        header: {
27          Accept: 'application/json',
28          'Content-Type': 'application/json',
29        },
30      }).then((response) => response.json()).then((json
31 ) => {
32        favorites.innerHTML = '';
33
34        // Affiche les favoris
35        if (json.animes.length > 0) {
36          json.animes.forEach((anime) => {
37            const favoriteItem = document.
38              createElement('div');
39            const favoritePicture = document.
40              createElement('img');
41            const favoriteRemover = document.
42              createElement('img');
43
44            favoriteItem.classList.add('favorite-item'
45 );
46            favoriteItem.dataset.id = anime.id;
47
48            favoriteItem.appendChild(favoritePicture);
49            favoriteItem.appendChild(favoriteRemover);
50
51            favorites.appendChild(favoriteItem);
52
53          });
54        }
55      });
56    }
57  }

```

```

39
40             if (col === true) {
41                 favoriteItem.classList.add('col-xl-4'
42             }
43
44             favoritePicture.src = anime.picture;
45             favoriteRemover.classList.add('favorite-
46             remover');
47             if (canDelete === false) {
48                 favoriteRemover.classList.add('d-none'
49             }
50             favoriteRemover.addEventListener('click'
51             , () => {
52                 fetch('/set/favorite', {
53                     method: 'PATCH',
54                     headers: {
55                         Accept: 'application/json',
56                         'Content-Type': 'application/
57                         json',
58                     },
59                     body: JSON.stringify({ idAnime:
60                         anime.id }),
61                     }).then((response) => response.json
62                     ).then(() => {
63                         fetchFavorites(false, true);
64                     });
65                     favoriteRemover.src = '/static/img/remove.
66                     svg';
67                     favoriteItem.appendChild(favoritePicture);
68                     favoriteItem.appendChild(favoriteRemover);
69                     favorites.appendChild(favoriteItem);
70                     });
71             } else {
72                 const noFavorite = document.createElement('div
73                     ');
74                 const h4 = document.createElement('h4');
75                 const span = document.createElement('span');
76
77                 noFavorite.classList.add('no-favorites');
78                 h4.innerHTML = 'Aucun favoris pour le moment';
79                 span.innerHTML = '??????';
80
81                 noFavorite.appendChild(h4);
82                 noFavorite.appendChild(span);

```

```
77             favorites.append(noFavorite);
78         }
79     });
80 }
81 }
82
```

```

1 /**
2  * Script pour la prise en charge des activités de l'
3  * utilisateur
4  *
5  * @author: Tanguy Cavagna
6  * @copyright: Copyright 2020, TPI
7  * @version: 1.0.0
8  * @date: 2020-06-02
9  */
10 /**
11  * Retourne le nombre d'heure, minutes et secondes entre une
12  * date et aujourd'hui
13  *
14  * @param {timestamp} olderTime Timestamp antérieur à aujourd'hui
15  */
16 /* eslint-disable-next-line no-unused-vars */
17 function timeElapsedFromNow(olderTime) {
18     let timeElapsed = 0;
19
20     // Récupération des différentes mesures de temps
21     const delta = Math.abs((Date.now() - olderTime) / 1000);
22     const seconds = Math.floor(delta % 60);
23     const hours = Math.floor(delta / 3600) % 24;
24     const minutes = Math.floor(delta / 60) % 60;
25
26     // Attribution de la mesure de temps la plus appropriée
27     if (hours > 0) {
28         timeElapsed = `${hours} heures`;
29     } else if (minutes > 0) {
30         timeElapsed = `${minutes} minutes`;
31     } else {
32         timeElapsed = `${seconds} secondes`;
33     }
34
35     return `Il y a ${timeElapsed}`;
36 }
37 /**
38  * Récupère les activités de l'utilisateur
39  */
40 /* eslint-disable-next-line no-unused-vars */
41 function fetchActivities() {
42     const activities = document.querySelector('.activities');
43
44     if (activities != null) {

```

```
45      const url = '/get/activities';
46
47      fetch(url, {
48          method: 'GET',
49          headers: {
50              Accept: 'application/json',
51              'Content-Type': 'application/json',
52          },
53      }).then((response) => response.json()).then((json
) => {
54          activities.innerHTML = '';
55
56          // Affiche les activités
57          if (json.activities.length > 0) {
58              json.activities.forEach((activity) => {
59                  const activityContainer = document.
60                  createElement('div');
60                  const wrapper = document.createElement('
61                  div');
61                  const activitySubject = document.
62                  createElement('div');
62                  const subjectTitle = document.
63                  createElement('span');
63                  const activityTimestamp = document.
64                  createElement('span');
64                  const activityImg = document.createElement
('img');
65
66                  activityContainer.classList = 'activity';
67                  subjectTitle.innerHTML = activity.title;
68                  activitySubject.classList.add('
activity__subject');
69                  activitySubject.appendChild(subjectTitle);
70                  activitySubject.innerHTML += `<br> À été
ajouté à : ${activity.list}`;
71                  activityTimestamp.classList.add('
activity__timestamp');
72                  activityTimestamp.innerHTML =
timeElapsedFromNow(new Date(activity.date));
73                  activityImg.src = activity.picture;
74
75                  wrapper.appendChild(activitySubject);
76                  wrapper.appendChild(activityTimestamp);
77                  wrapper.appendChild(activityImg);
78                  activityContainer.appendChild(wrapper);
79                  activities.appendChild(activityContainer);
80              });
}
```

```
81          } else {
82              const noFavorites = document.createElement('
83                  div');
84
85                  const h4 = document.createElement('h4');
86                  const span = document.createElement('span');
87
88                  noFavorites.className = 'no-activities';
89                  h4.innerHTML = 'Aucune activité pour le
90                      moment';
91
92                  span.innerHTML = '????????';
93
94                  noFavorites.appendChild(h4);
95                  noFavorites.appendChild(span);
96
97                  activities.appendChild(noFavorites);
98          }
99      );
100  }
101 }
```

```

1  /**
2   * Script pour la prise en charge du changement d'ordre des
3   * favoris
4   *
5   * @author: Tanguy Cavagna
6   * @copyright: Copyright 2020, TPI
7   * @version: 1.0.0
8   * @date: 2020-06-02
9   */
10 const reorder = document.querySelector('.reorder');
11 let removers = document.querySelectorAll('.favorite-remover');
12 let toggleReorder = false;
13
14 /**
15  * Affiche ou cache les boutons de suppression des favoris
16 */
17 function toggleHideFavoriteRemovers(hide = true) {
18     removers.forEach((remover) => {
19         if (hide === true) {
20             remover.classList.add('d-none');
21         } else {
22             remover.classList.remove('d-none');
23         }
24     });
25 }
26
27 reorder.addEventListener('click', () => {
28     $('.favorites').sortable({ disabled: toggleReorder });
29     reorder.innerHTML = toggleReorder === false ? 'Sauvegarder'
30       : 'Réordonner les favoris';
31
32     // Si bouton cliqué pour sauvegarder
33     if (toggleReorder === true) {
34         const favorites = document.querySelectorAll('..
35         favorites .favorite-item');
36         toggleHideFavoriteRemovers(true);
37
38         const ids = [];
39         favorites.forEach((favorite) => {
40             ids.push(favorite.dataset.id);
41         });
42
43         fetch('/set/favorites-order', {
44             method: 'PATCH',
45             headers: {
46                 Accept: 'application/json',

```

```
45          'Content-Type' : 'application/json',
46          },
47          body: JSON.stringify({ ids }),
48        }).then((response) => response.json()).then(() => {
49          // Ne rien faire
50        }).catch(() => {
51          // Ne rien faire
52        });
53    } else { // Clique sur `Réordonner les favoris` pour
      afficher les boutons de suppression
54      removers = document.querySelectorAll('.favorite-
      remover');
55      toggleHideFavoriteRemovers(false);
56    }
57
58    toggleReorder = !toggleReorder;
59  });
60
```

```

1 /**
2  * Script pour la prise en charge des listes de l'utilisateur
3 *
4  * @author: Tanguy Cavagna
5  * @copyright: Copyright 2020, TPI
6  * @version: 1.0.0
7  * @date: 2020-06-02
8 */
9
10 let lists = document.querySelectorAll('#lists .lists-container ul li');
11 let listNames = document.querySelectorAll('#lists .lists-container ul li .list__name');
12 let listRemovables = document.querySelectorAll('#lists .lists-container ul li .remove_list');
13 let listCancelRenames = document.querySelectorAll('#lists .lists-container ul li .cancel_rename');
14 let listRenameInput = document.querySelectorAll('#lists .lists-container ul li input');
15 const listSearchString = document.querySelector('#search-list .search-string');
16 const newListString = document.querySelector('#new-list__name');
17
18 /**
19  * Récupère tout les animes d'une liste et les affiche
20 *
21 * @param {string} searchTerms Chaine de recherche
22 * @param {int} idList Id de la liste à fetch
23 */
24 function fetchListAnimes(searchTerms, idList) {
25   fetch('/get/animes', {
26     method: 'POST',
27     headers: {
28       Accept: 'application/json',
29       'Content-Type': 'application/json',
30     },
31     body: JSON.stringify({
32       nicknameUser: document.getElementById('
33       searched_user').dataset.nickname,
34       idList,
35       'search-terms': searchTerms,
36     }),
37   }).then((response) => response.json()).then((json) => {
38     const { animes } = json; // Récupère la valeur de la
      clef `animes` de l'object `json`

```

```
39         const animesContainer = document.querySelector('#  
 40           animes-container');  
 41       if (Object.keys(animes).length > 0) {  
 42         const animeLists = Object.keys(animes);  
 43  
 44         animesContainer.innerHTML = '';  
 45  
 46         animeLists.forEach((list) => {  
 47           const listSection = document.createElement('  
 48             div');  
 49           const listSectionTitle = document.  
 50             createElement('h2');  
 51           listSection.classList = 'mb-5 list-section';  
 52           listSectionTitle.innerHTML = list;  
 53  
 54           animes[list].forEach((anime) => {  
 55             const animeItem = document.createElement('  
 56               div');  
 57             const animeItemTitle = document.  
 58               createElement('h4');  
 59             const animeItemPicture = document.  
 60               createElement('img');  
 61  
 62             animeItem.classList.add('anime-item');  
 63             animeItemTitle.innerHTML = anime.title;  
 64             animeItemPicture.src = anime.picture;  
 65             animeItem.appendChild(animeItemTitle);  
 66             animeItem.appendChild(animeItemPicture);  
 67  
 68             listSection.appendChild(animeItem);  
 69           });  
 70         animesContainer.appendChild(listSectionTitle);  
 71         animesContainer.appendChild(listSection);  
 72       }  
 73     } else {  
 74       animesContainer.innerHTML = '';  
 75  
 76       const infos = document.createElement('div');  
 77       infos.classList.add('lists-empty');  
 78  
 79       const title = document.createElement('h4');  
 80       const separator = document.createElement('span');  
 81       const sub = document.createElement('span');
```

```

80          const img = document.createElement('img');
81          title.innerHTML = 'Aucun anime';
82          separator.classList.add('separator');
83          sub.innerHTML = 'Remplis tes listes pour la
rendre heureuse !';
84          img.src = '/static/img/whale.svg';
85
86          infos.appendChild(title);
87          infos.appendChild(separator);
88          infos.appendChild(sub);
89          infos.appendChild(img);
90
91          animesContainer.appendChild(infos);
92      }
93  });
94 }
95
96 /**
97 * Récupère les animes de la list cliquée
98 */
99 function listNamesClickEventSetup() {
100     listNames.forEach((listName) => {
101         const currentListName = listName;
102
103         // Event clique pour le nom des listes afin d'
afficher les animes de cette dite liste
104         currentListName.addEventListener('click', () => {
105             const list = currentListName.parentElement;
106
107             if (!list.classList.contains('new-list')) {
108                 // Parcour toutes les listes existantes dans
la page et supprime la classe
109                 // `selected` pour l'ajouté sur la liste
courante
110                 lists.forEach((l) => l.classList.remove(
'selected'));
111                 list.classList.add('selected');
112
113                 fetchListAnimes(listSearchString.value, list.
dataset.list);
114             }
115         });
116
117         if (
118             ![
119                 'Complétés',
120                 'En cours',

```

```

121          'Planifiés',
122          'Abandonés',
123      ].includes(currentListName.innerHTML)
124  ) {
125      // Event double clique pour renommer la liste
126      // seulement si elle ne fait
127      // pas parti de celles par défaut
128      currentListName.addEventListener('dblclick'
129      , () => {
130          const newNameInput = currentListName.
131          parentNode.querySelector('input');
132          const cancelButton = currentListName.
133          parentNode.querySelector('.cancel__rename');
134          const removeButton = currentListName.
135          parentNode.querySelector('.remove__list');
136
137          newNameInput.classList.remove('d-none');
138          newNameInput.value = currentListName.
139          innerHTML;
140          cancelButton.classList.remove('d-none');
141          currentListName.classList.add('d-none');
142          removeButton.classList.add('d-none');
143      });
144  );
145
146 /**
147 * Supprime une liste
148 *
149 * @param {int} idList Id de la liste à supprimer
150 */
151 function deleteList(idList) {
152     fetch('/delete/list', {
153         method: 'DELETE',
154         headers: {
155             Accept: 'application/json',
156             'Content-Type': 'application/json',
157         },
158         body: JSON.stringify({
159             idList,
160         }),
161     }).then((response) => response.json()).then(() => {
162         window.location.reload();
163     });
164 }
165

```

```

162 /**
163 * Supprime la liste cliquée
164 */
165 function removeListClickEventSetup() {
166     // Event clique pour la suppression d'une liste
167     listRemovables.forEach((removeTrigger) => {
168         removeTrigger.addEventListener('click', () => {
169             const list = removeTrigger.parentNode;
170             deleteList(list.dataset.list);
171         });
172     });
173 }
174
175 /**
176 * Annule le renommage de la liste
177 */
178 function cancelListRenameClickEvent() {
179     // Event clique pour l'annulation du renommage d'une
      liste
180     listCancelRenames.forEach((cancelTrigger) => {
181         cancelTrigger.addEventListener('click', () => {
182             const list = cancelTrigger.parentNode;
183             const newNameInput = list.querySelector('input');
184             const cancelButton = list.querySelector('.cancel__rename');
185             const removeButton = list.querySelector('.remove__list');
186             const listName = list.querySelector('.list__name');
187
188             newNameInput.classList.add('d-none');
189             newNameInput.value = '';
190             cancelButton.classList.add('d-none');
191             listName.classList.remove('d-none');
192             removeButton.classList.remove('d-none');
193         });
194     });
195 }
196
197 /**
198 * Ajoute une nouvelle liste personnelle à l'utilisateur
199 *
200 * @param {string} newListName Nom de la nouvelle liste
201 */
202 function addNewList(newListName) {
203     fetch('/add/list', {
204         method: 'PUT',

```

```

205     headers: {
206         Accept: 'application/json',
207         'Content-Type': 'application/json',
208     },
209     body: JSON.stringify({
210         newListName,
211     }),
212 }).then((response) => response.json()).then((json) => {
213     newListString.value = '';
214     const createdList = json.list;
215
216     const li = document.createElement('li');
217     const listName = document.createElement('span');
218     const updateNameInput = document.createElement('input')
219     );
220     const removeButton = document.createElement('span');
221
222     li.dataset.list = createdList.id;
223     listName.classList.add('list__name');
224     listName.innerHTML = createdList.name;
225     updateNameInput.type = 'text';
226     updateNameInput.name = 'new_list_name';
227     updateNameInput.style.width = '80%';
228     updateNameInput.classList.add('d-none');
229     updateNameInput.id = `${createdList.id}_new_list_name
`;;
230     removeButton.classList.add('remove__list');
231     removeButton.innerHTML = '!';
232
233     li.appendChild(listName);
234     li.appendChild(updateNameInput);
235     li.appendChild(removeButton);
236
237     // Ajoute la nouvelle liste juste avant le champ text
238     // permettant d'ajouter
239     // une nouvelle liste
240     document.querySelector('#lists .lists-container ul').
241     insertBefore(li, lists[lists.length - 1]);
242
243     // Mise à jour des éléments DOM pour les listes
244     lists = document.querySelectorAll('#lists .lists-
245     container ul li');
246     listNames = document.querySelectorAll('#lists .lists-
247     container ul li .list__name');
248     listRemovables = document.querySelectorAll('#lists .
249     lists-container ul li .remove__list');
250     listCancelRenames = document.querySelectorAll('#lists

```

```

244 .lists-container ul li .cancel__rename');
245     listRenameInput = document.querySelectorAll('#lists .
  lists-container ul li input');
246
247     // Ajout des events pour la liste venant d'être créer
248     listNamesClickEventSetup();
249     removeListClickEventSetup();
250     cancellListRenameClickEvent();
251   });
252 }
253
254 /**
255 * Renomme la liste
256 *
257 * @param {int} listId Id de la liste à modifier
258 * @param {string} newListName Nouveau nom de la liste
259 */
260 function renameList(idList, newListName) {
261   fetch('/set/list/name', {
262     method: 'PATCH',
263     headers: {
264       Accept: 'application/json',
265       'Content-Type': 'application/json',
266     },
267     body: JSON.stringify({
268       idList,
269       newListName,
270     }),
271   }).then((response) => response.json()).then(() => {
272     window.location.reload();
273   });
274 }
275
276 // Affiche les animes des listes de l'utilisateur dont le
  titre concorde avec
277 // la chaine recherchée lorsque la touche entrés est pressée
278 listSearchString.onkeydown = (e) => {
279   if (e.keyCode === 13) { // Touche entrée
280     fetchListAnimes(listSearchString.value, null);
281   }
282 };
283
284 // Créer une nouvelle liste pour l'utilisateur connecté
  lorsque la touche entrée et pressée
285 if (newListString !== undefined) {
286   newListString.onkeydown = (e) => {
287     if (e.keyCode === 13) {

```

```
288         addNewList(newListString.value);
289     }
290   };
291 }
292
293 // Renomme une liste
294 if (listRenameInput !== undefined) {
295   listRenameInput.forEach(renameInput) => {
296     const currentRenameInput = renameInput;
297
298     if (currentRenameInput.id !== 'new-list__name') {
299       currentRenameInput.onkeydown = (e) => {
300         if (e.keyCode === 13) {
301           renameList(
302             currentRenameInput.parentNode.dataset
303               .list,
304               currentRenameInput.value,
305             );
306           };
307         };
308       });
309     }
310
311 window.addEventListener('load', () => {
312   fetchListAnimes(null, null);
313   listNamesClickEventSetup();
314   removeListClickEventSetup();
315   cancelListRenameClickEvent();
316 });
317
```

```
1 /**
2  * Script pour la prise en charge de la page d'accueil lorsque
3  * l'utilisateur connecté ou pas
4  *
5  * @author: Tanguy Cavagna
6  * @copyright: Copyright 2020, TPI
7  * @version: 1.0.0
8  * @date: 2020-05-26
9  */
10 const background = document.querySelector('.background');
11 const navbar = document.querySelector('.navbar');
12
13 if (background != null) {
14     navbar.style.background = 'transparent';
15 }
16
```

```
1  /**
2   * @filesource style.scss
3   * @brief Fichier contenant tout le style de mon application
4   * @author Tanguy Cavagna <tanguy.cvgn@eduge.ch>
5   * @date 2020-04-18
6   * @version 1.0.0
7  */
8
9 @import 'typography';
10 @import 'colors';
11 @import 'sizes';
12
13 // Global
14 //=====
15 *{
16     font-family: $f-regular;
17     color: $c-white;
18 }
19 body {
20     background: $c-background-medium-dark;
21 }
22 input {
23     background: $c-background-darkest;
24     color: $c-light-gray;
25     border: none;
26     padding: 10px;
27     border-radius: $s-small-border-radius;
28 }
29
30 // Scrollbar
31 //=====
32 // Largeur
33 ::-webkit-scrollbar {
34     width: 10px;
35 }
36 // Fond
37 ::-webkit-scrollbar-track {
38     background: $c-background-darker;
39 }
40 // Barre
41 ::-webkit-scrollbar-thumb {
42     background: $c-background-medium-dark;
43 }
44 // Hover de la barre
45 ::-webkit-scrollbar-thumb:hover {
46     background: $c-background-darkest;
47 }
```

```
48
49 // Navbar
50 //=====
51 .navbar {
52     background: $c-background-dark;
53     height: $s-nav-height;
54 }
55 .nav-link {
56     color: $c-white;
57     font-family: $f-nav-link;
58     letter-spacing: .15rem;
59     font-size: .9rem;
60     margin-right: 20px;
61
62     &:hover {
63         color: #c3c3c3;
64     }
65 }
66 .dropdown-item span.danger {
67     padding: .25rem .5rem;
68     background-color: $c-danger-background;
69     color: $c-danger-foreground;
70     border-radius: 5px;
71 }
72 .navbar-toggler:not(:disabled) ~ .navbar-collapse.show,
73 .navbar-toggler:not(:disabled) ~ .navbar-collapse.collapsing {
74     margin-top: 15px;
75 }
76 .navbar-toggler-icon {
77     width: auto;
78     height: auto;
79     display: flex;
80     justify-content: center;
81     align-items: center;
82 }
83
84 // Login & Register
85 //=====
86 .login-form {
87     width: $s-login-form-width;
88     height: $s-login-form-height;
89     padding-bottom: 60px;
90 }
91 .signup-form {
92     width: $s-register-form-width;
93     height: $s-register-form-height;
94     padding-bottom: 60px;
```

```
95 }
96 .login-form,
97 .signup-form {
98     background: $c-background-darker;
99     border-radius: $s-small-border-radius;
100    display: flex;
101   flex-direction: column;
102   align-items: center;
103 }
104 .login-form h3,
105 .signup-form h3 {
106   color: $c-light-gray;
107   font-family: $f-section-title;
108   padding-top: 60px;
109   padding-bottom: 60px;
110   text-align: center;
111 }
112 .login-form form,
113 .signup-form form {
114   height: 100%;
115   width: 80%;
116   display: flex;
117   flex-direction: column;
118
119   fieldset {
120     width: 100%;
121   }
122   ul {
123     list-style-type: none;
124     background-color: $c-danger-background;
125     padding: 5px 0 5px 0;
126     display: flex;
127     flex-direction: column;
128     align-items: center;
129     border-radius: $s-small-border-radius;
130   }
131   ul li {
132     color: $c-danger-foreground;
133     font-family: $f-regular;
134     font-weight: 600;
135   }
136   input {
137     margin-bottom: 7%;
138     width: 100%;
139     font-family: $f-regular;
140     font-weight: bold;
141 }
```

```
142     .submit {
143         margin-top: 5%;
144         display: flex;
145         flex-direction: column;
146         align-items: center;
147
148         button {
149             width: auto;
150             font-size: 1.3rem;
151             border: none;
152             border-radius: $s-small-border-radius;
153             background: $c-accent;
154             color: $c-white;
155             padding: 5px 20px 5px 20px;
156             margin-bottom: 15px;
157         }
158
159         span {
160             color: $c-white;
161         }
162     }
163 }
164
165 // Search modal
166 //=====
167 .search-modal {
168     margin-top: 4%;
169
170     .modal-dialog {
171         max-width: none;
172         width: 40%;
173     }
174 }
175 .search-form {
176     display: flex;
177     flex-direction: row;
178     background: $c-background-medium-dark;
179
180     div:first-child {
181         position: relative;
182         margin-left: 10px;
183
184         img {
185             position: absolute;
186             top: 50%;
187             left: 50%;
188             transform: translateY(-50%);
```

```
189         width: 20px;
190         opacity: .4;
191     }
192 }
193 div:last-child {
194     position: relative;
195     margin-left: 10px;
196
197     svg {
198         position: absolute;
199         top: 50%;
200         left: 50%;
201         transform: translateY(-50%);
202         width: 20px;
203         opacity: .4;
204     }
205 }
206 input {
207     width: 85%;
208     margin-left: 30px;
209     font-family: $f-regular;
210     font-weight: bold;
211 }
212 }
213 .search-string {
214     background-color: $c-background-medium-dark;
215 }
216 .search-clear:hover {
217     cursor: pointer;
218 }
219
220 // Home
221 //=====
222 #override-info {
223     position: absolute;
224     top: 90px;
225     left: 50%;
226     transform: translateX(-50%);
227     color: $c-white;
228     width: 40%;
229     padding: 0;
230     background: $c-background-darker;
231     border-radius: $s-small-border-radius;
232     text-align: center;
233     z-index: 10;
234 }
235 #overrided {
```

```
236     position: absolute;
237     left: 50%;
238     width: 40%;
239     transform: translateX(-50%);
240     top: 90px;
241
242     button {
243         margin: auto;
244     }
245 }
246 .main-home {
247     max-width: 100%;
248
249     section {
250         padding-left: 80px;
251
252         h4 {
253             font-family: $f-section-title;
254             letter-spacing: .3rem;
255             font-size: 1.7rem;
256             margin: auto;
257             text-align: center;
258             margin-bottom: 30px;
259         }
260         .search-results {
261             display: grid;
262             grid-template-columns: repeat(auto-fill, 350px);
263             column-gap: 50px;
264             row-gap: 20px;
265
266             .result-item {
267                 display: flex;
268                 position: relative;
269                 height: auto;
270
271                 .left, .right {
272                     flex: 50%;
273                     position: relative;
274                 }
275                 .left {
276                     .picture {
277                         width: 100%;
278                         height: calc(100% * 0.73333);
279                         border-radius: $s-small-border-radius
280                         0 0 $s-small-border-radius;
281                     }
282                     .title {
```

```
282             position: absolute;
283             left: 0;
284             width: 100%;
285             bottom: 0;
286             text-align: center;
287             background: rgba($color: $c-
background-dark, $alpha: .8);
288             border-radius: 0 0 0 $s-small-border-
radius;
289             font-size: .9rem;
290             padding: .5rem;
291
292             &:hover {
293                 cursor: pointer;
294             }
295         }
296     }
297     .right {
298         .status,
299         .type,
300         .relations,
301         .episodes {
302             width: 100%;
303             min-height: 30px;
304             text-align: center;
305             display: flex;
306             justify-content: center;
307             align-items: center;
308         }
309         .status {
310             border-radius: 0 $s-small-border-
radius 0 0;
311             background: $c-background-darkest;
312             color: $c-primary-light;
313         }
314         .type {
315             background: $c-background-darkest--;
316             color: $c-light-gray;
317         }
318         .relations {
319             justify-content: space-between;
320             align-items: flex-start;
321             flex-wrap: wrap;
322             background: $c-background-darker;
323             max-height: calc(100% - (3 * 30px));
324             min-height: calc(100% - (3 * 30px));
325             overflow-x: hidden;
```

```
326                     overflow-y: auto;
327                     padding: 15px;
328                     margin-left: 0;
329
330                     img {
331                         border-radius: $s-small-border-
332                             radius;
333                         margin-bottom: 15px;
334                         padding: 0;
335                     }
336                     .episodes {
337                         border-radius: 0 0 $s-small-border-
338                             radius 0;
339                         background: $c-background-darkest--;
340                         color: $c-light-gray;
341                     }
342                     .modal .modal-dialog {
343                         max-width: none;
344                         width: 50%;
345
346                     .modal-content {
347                         background: $c-background-darkest--;
348
349                     .header {
350                         position: relative;
351                         height: 170px;
352                         width: 100%;
353
354                     .header-background {
355                         height: 100%;
356
357                         img {
358                             top: 0;
359                             left: 0;
360                             width: 100%;
361                             height: 100%;
362                             object-fit: cover;
363                             border-radius: $s-small-
364                             border-radius $s-small-border-radius 0 0;
365                         }
366                     .dark-layer {
367                         top: 0;
368                         left: 0;
369                         height: 100%;
```

```
370                                     background: rgba($color:  
371                                         #000000, $alpha: .5);  
372                                     border-radius: $s-small-  
373                                         border-radius $s-small-border-radius 0 0;  
374                                         }  
375                                         }  
376                                         .thumbnail,  
377                                         .dark-layer,  
378                                         .title,  
379                                         .favorite-toggler,  
380                                         .save {  
381                                         position: absolute;  
382                                         }  
383                                         .thumbnail {  
384                                         left: 5%;  
385                                         top: 20%;  
386                                         width: 140px;  
387                                         border-radius: $s-small-  
388                                         border-radius;  
389                                         }  
390                                         .title {  
391                                         left: 25%;  
392                                         bottom: 5%;  
393                                         max-width: 400px;  
394                                         }  
395                                         .favorite-toggler {  
396                                         bottom: 7%;  
397                                         right: 2%;  
398                                         }  
399                                         .favorite-toggler:hover {  
400                                         cursor: pointer;  
401                                         }  
402                                         .favorite-toggler:hover svg path,  
403                                         .favorite-toggler.is-favorite svg  
404                                         path {  
405                                         fill: $c-favorite;  
406                                         }  
407                                         .save {  
408                                         right: 2%;  
409                                         bottom: 2%;  
410                                         background: $c-primary-light;  
411                                         color: $c-white;  
412                                         }  
413                                         }  
414                                         .content {  
415                                         padding: 80px 5% 0 5%;  
416                                         min-height: 200px;
```

```
413 height: auto;
414 background: $c-background-darkest
415 --;
416 border-radius: 0 0 $s-small-
417 border-radius $s-small-border-radius;
418
419 fieldset {
420 width: 100%;
421 display: flex;
422 flex-direction: column;
423 align-items: flex-start;
424 margin-bottom: 20px;
425
426 & > span:first-child {
427 margin-bottom: 10px;
428 font-family: $f-regular;
429 font-weight: bold;
430 letter-spacing: .2rem;
431 }
432
433 .status {
434 select {
435 background: $c-background
436 -medium-dark;
437 color: $c-light-gray;
438 padding: 5px 0 5px 15px;
439 width: 30%;
440 outline: none;
441 border: none;
442 border-radius: 5px;
443 margin-left: 20px;
444
445 option:hover {
446 background: $c-
447 background-darkest;
448 }
449 }
450 }
451
452 .custom .lists {
453 width: 100%;
454 display: flex;
455 flex-direction: column;
456 flex-wrap: wrap;
457 max-height: 130px;
458 overflow-x: auto;
459 padding-left: 20px;
460
461 }
```

```
456          .list-item {
457              display: flex;
458              margin-bottom: 10px;
459              min-width: 300px;
460
461          .list-title {
462              font-size: .9rem;
463          }
464      }
465  }
466  }
467  }
468  }
469  }
470 }
471 .activities {
472     margin-top: 5%;
473     padding-top: 5%;
474     height: auto;
475     border-top: 3px solid $c-background-darkest;
476     display: grid;
477     grid-template-columns: repeat(auto-fill, minmax(
478         300px, 1fr));
479     gap: 30px;
480     margin-bottom: 50px;
481
482     .activity {
483         width: 100%;
484         height: 200px;
485         position: relative;
486         overflow: hidden;
487         display: flex;
488         justify-content: center;
489         align-items: center;
490
491         div {
492             width: 100%;
493             z-index: 0;
494
495             img {
496                 position: absolute;
497                 top: 0;
498                 left: 50%;
499                 transform: translateX(-50%);
500                 width: 100%;
501                 opacity: .3;
502                 z-index: -1;
```

```
502          }
503          .activity__subject {
504              width: 100%;
505              text-align: center;
506              z-index: 1;
507              background-color: rgba($color: $c-
background-darker, $alpha: .7);
508
509          span {
510              color: $c-primary-light;
511              font-weight: bold;
512              font-size: 1.8rem;
513          }
514      }
515      .activity__timestamp {
516          z-index: 1;
517          position: absolute;
518          top: 5%;
519          right: 5%;
520          font-family: $f-nav-link;
521          color: $c-light-gray;
522      }
523  }
524 }
525 .no-activities {
526     width: 100%;
527     display: flex;
528     justify-content: center;
529     align-items: center;
530     flex-direction: column;
531
532     h4, span {
533         color: $c-light-gray;
534     }
535     span {
536         font-weight: bold;
537         font-size: 4rem;
538         opacity: .5;
539     }
540 }
541 }
542 }
543 aside {
544     padding: 0 80px;
545
546     h4 {
547         padding-left: 30px;
```

```
548         color: $c-light-gray;
549         font-family: $f-nav-link;
550         font-size: .9rem;
551         letter-spacing: .25rem;
552         margin-bottom: 20px;
553     }
554 }
555 }
556
557 // About
558 //=====
559 .about-background {
560     opacity: .2;
561     position: fixed;
562     z-index: -1;
563     top: 50%;
564     left: 50%;
565     transform: translate(-50%, -50%);
566 }
567 .about-content {
568     z-index: 1;
569     text-align: justify;
570
571     h1 {
572         font-family: $f-nav-link;
573         text-transform: uppercase;
574     }
575     h2 {
576         font-weight: bold;
577     }
578     h2 + h3 {
579         margin-top: 0;
580     }
581     h3 {
582         margin-top: 15%;
583     }
584     img {
585         max-width: 100%;
586     }
587 }
588
589 // Profile
590 //=====
591 .profile-header {
592     .header {
593         position: relative;
594         height: 40vh;
```

```
595
596     .banner,
597     .overlay {
598         position: absolute;
599         width: 100%;
600         height: 100%;
601         left: 0;
602         top: 0;
603         object-fit: cover;
604     }
605     .overlay {
606         background: linear-gradient(180deg, rgba(196, 196
607 , 196, 0) 0%, rgba(0, 0, 0, 0.55) 100%);
608     }
609     .profile-picture {
610         position: absolute;
611         width: 10%;
612         min-width: 125px;
613         height: auto;
614         left: 5%;
615         bottom: 0;
616     }
617     .profile-nickname {
618         position: absolute;
619         bottom: 40px;
620         font-family: $f-nav-link;
621         font-size: 1.8rem;
622     }
623     .links {
624         height: 6vh;
625         background: $c-background-dark;
626         padding-left: 17%;
627         display: flex;
628         align-items: center;
629
630         span {
631             margin-right: 45px;
632             font-family: $f-nav-link;
633             color: $c-white;
634
635             &:hover {
636                 cursor: pointer;
637                 color: $c-light-gray;
638             }
639             a {
640                 color: inherit;
```

```
641             font-family: inherit;
642             text-decoration: none;
643         }
644     }
645 }
646 }
647 .profile {
648     display: flex;
649     flex-direction: column;
650     align-items: flex-end;
651
652     .stats {
653         width: 95%;
654         background: $c-background-dark;
655         border-radius: 10px;
656         display: flex;
657         flex-wrap: wrap;
658         justify-content: left;
659         align-items: center;
660         position: relative;
661         padding: 30px;
662         margin: auto;
663         margin-top: 20px;
664         margin-bottom: 20px;
665         overflow-x: auto;
666     }
667     .favorites-container {
668         width: 95%;
669         height: auto;
670         margin: auto;
671         background: $c-background-dark;
672         overflow-x: auto;
673     }
674     .stats {
675         display: flex;
676         align-items: center;
677         justify-content: space-around;
678
679         .watched {
680             display: flex;
681             flex-direction: column;
682
683             span {
684                 text-align: center;
685             }
686             .count,
687             .category {
```

```
688         font-family: $f-nav-link;
689     }
690     .count {
691         color: $c-accent;
692         font-size: 2rem;
693     }
694 }
695 }
696 }
697
698 // Lists
699 //=====
700 #lists {
701     display: flex;
702     justify-content: center;
703
704     .lists-container {
705         width: 80%;
706
707         #search-list {
708             display: flex;
709             justify-content: left;
710             align-items: center;
711             background: $c-background-darkest;
712             border-radius: $s-small-border-radius;
713
714             img { margin-left: 5px; }
715             .search-string {
716                 width: calc(100% - 25px);
717                 background: inherit;
718             }
719         }
720         h4 {
721             margin-top: 5%;
722             font-size: 1.3rem;
723             font-weight: bold;
724         }
725         ul {
726             list-style-type: none;
727             padding-left: 20px;
728
729             li {
730                 padding: 5px;
731                 display: flex;
732                 justify-content: space-between;
733
734                 & .list__name:hover,
```

```
735             & .remove__list:hover,
736             & .cancel__rename:hover {
737                 cursor: pointer;
738             }
739             &.selected {
740                 background: $c-background-darker;
741                 border-radius: $s-small-border-radius;
742             }
743         }
744     }
745 }
746 #animes-container {
747     h2 {
748         width: 100%;
749     }
750     .list-section {
751         display: flex;
752         justify-content: left;
753         flex-wrap: wrap;
754
755         .anime-item {
756             position: relative;
757             display: flex;
758             justify-content: center;
759             height: auto;
760             width: 200px;
761             margin: 25px;
762
763             h4 {
764                 position: absolute;
765                 left: 0;
766                 width: 100%;
767                 bottom: 0;
768                 text-align: center;
769                 background: rgba($color: $c-background-dark,
770 $alpha: .8);
771                 border-radius: 0 0 0 $s-small-border-radius;
772                 font-size: .9rem;
773                 padding: .5rem;
774                 margin: 0;
775             }
776             img {
777                 width: 100%;
778                 height: calc(200px / .73333);
779                 border-radius: $s-small-border-radius;
780             }
781 }
```

```
781         }
782     }
783 }
784 .lists-empty img {
785     margin-top: 30px;
786 }
787 .new-list #new-list__name {
788     width: 100%;
789 }
790
791 // Favorites
792 =====
793 .reorder {
794     margin-top: 20px;
795     margin-right: 2.5%;
796     padding: 5px;
797     border: none;
798     width: 12%;
799     background: $c-primary;
800     border-radius: $s-small-border-radius;
801
802     &:focus {
803         outline: none;
804     }
805 }
806 .favorites-container {
807     background: $c-background-darker;
808     border-radius: 10px;
809     width: 100%;
810     height: 750px;
811 }
812 .favorites {
813     padding: 20px;
814     position: relative;
815     display: grid;
816     grid-template-columns: repeat(auto-fill, 90px);
817     grid-template-rows: repeat(auto-fill, 121px);
818     gap: 20px;
819
820     .no-favorites {
821         display: flex;
822         flex-direction: column;
823         align-items: center;
824         position: absolute;
825         top: 50%;
826         left: 50%;
827         transform: translateX(-50%) translateY(-50%);
```

```
828         width: 100%;  
829  
830     h4 {  
831         font-family: $f-section-title;  
832         letter-spacing: .3rem;  
833         font-size: 1.7rem;  
834         text-align: center;  
835         padding: 0 1.2rem;  
836     }  
837     span {  
838         font-family: $f-nav-link;  
839         font-size: 2rem;  
840         margin-top: 5%;  
841         opacity: .5;  
842     }  
843 }  
844 .favorite-item {  
845     width: 100%;  
846     height: 100%;  
847     position: relative;  
848  
849     img {  
850         width: 100%;  
851         height: 100%;  
852         border-radius: 3px;  
853  
854     &.favorite-remover {  
855         position: absolute;  
856         right: -10px;  
857         top: -10px;  
858         width: 20px;  
859         height: 20px;  
860  
861         &:hover {  
862             cursor: pointer;  
863         }  
864     }  
865 }  
866 }  
867 }  
868  
869 // miscellaneous  
870 //=====  
871 .background {  
872     position: fixed;  
873     left: 0;  
874     top: 0;
```

```
875     z-index: -1;
876 }
877 .container-center {
878     display: flex;
879     flex-direction: column;
880     justify-content: center;
881     align-items: center;
882     width: 100%;
883     height: calc(100vh - #{$s-nav-height});
884 }
885 .alert button {
886     width: 16px;
887     height: 16px;
888     display: flex;
889     margin-top: 4px;
890     margin-left: 10px;
891 }
892 .alert-success {
893     color: $c-success-foreground;
894     background: $c-success-background;
895     font-weight: bold;
896     border: none;
897 }
898 input:focus {
899     outline: none;
900 }
901 .custom-check {
902     width: 20px;
903     height: 20px;
904     background: $c-light-gray;
905     border-radius: $s-small-border-radius;
906     display: inline-block;
907     margin-right: 10px;
908     position: relative;
909
910     input {
911         display: none;
912     }
913 }
914 .custom-check.checked::before {
915     content: '';
916     position: absolute;
917     top: 50%;
918     left: 50%;
919     transform: translateX(-50%) translateY(-50%);
920 }
921 .custom-check.checked {
```

```
922     background: $c-primary-light;
923     position: relative;
924 }
925 .no-search,
926 .lists-empty {
927     display: flex;
928     flex-direction: column;
929     align-items: center;
930
931     span,
932     h4 {
933         color: $c-light-gray;
934         margin: 0;
935     }
936     h4 {
937         font-family: $f-section-title;
938         letter-spacing: .3rem;
939         font-size: 1.7rem;
940     }
941     .separator {
942         width: 80px;
943         height: 3px;
944         background: $c-light-gray;
945         margin: 1.5rem 0;
946     }
947     span:nth-child(3) {
948         font-size: 1.2rem;
949         text-align: center;
950         font-family: $f-nav-link;
951         letter-spacing: .2rem;
952     }
953     span:nth-child(4) {
954         font-family: $f-nav-link;
955         font-size: 4rem;
956         opacity: .5;
957         margin-top: 5%;
958     }
959 }
960 .toc {
961     width: 15%;
962     height: auto;
963     position: fixed;
964     top: 100px;
965     left: 40px;
966
967     a {
968         color: $c-light-gray;
```

```
969      }
970      h1:hover, h2:hover, h3:hover, h4:hover {
971          cursor: pointer;
972      }
973      h1 {
974          font-size: 1.6rem;
975          font-weight: bold;
976      }
977      h2 {
978          font-size: 1.4rem;
979          margin-left: 10px;
980      }
981      h3 {
982          font-size: 1.1rem;
983          margin-left: 20px;
984      }
985      h4 {
986          font-size: 1rem;
987          margin-left: 30px;
988      }
989 }
990
991 // Media queries
992 //=====
993 @media (min-width: 1450px) {
994     .main-home aside .favorites {
995         max-height: 750px;
996         overflow-x: hidden;
997         overflow-y: auto;
998     }
999 }
1000 @media (max-width: 1449px) {
1001     .main-home section {
1002         padding-right: 80px;
1003     }
1004 }
1005 @media (min-width: 992px) {
1006     .profile-header .profile-nickname {
1007         left: 17%;
1008     }
1009 }
1010 @media (max-width: 991px) {
1011     .profile-header .profile-nickname {
1012         right: 17%;
1013     }
1014 }
```

```
1  /**
2   * @filesource _typography.scss
3   * @brief Fichier contenant toutes les tailles utilisées dans
4   * mon application
5   * @author Tanguy Cavagna <tanguy.cvgn@eduge.ch>
6   * @date 2020-04-18
7   * @version 1.0.0
8
9
10 // Navbar
11 //=====
12 $s-nav-height: 60px;
13
14 // Login
15 //=====
16 $s-login-form-width: 450px;
17 $s-login-form-height: auto;
18
19 // Register
20 //=====
21 $s-register-form-width: 450px;
22 $s-register-form-height: auto;
23
24 // Miscellaneous
25 //=====
26 $s-small-border-radius: 5px;
```

```
1  /**
2   * @filesource _typography.scss
3   * @brief Fichier contenant toutes les couleurs utilisées dans
4   * mon application
5   * @author Tanguy Cavagna <tanguy.cvgn@eduge.ch>
6   * @date 2020-04-18
7   * @version 1.0.0
8   */
9 $c-white: #fff;
10 $c-light-gray: #C4C4C4;
11 $c-primary-light: #1CA5F2;
12 $c-primary: #006EDB;
13 $c-background-medium-dark: #272C38;
14 $c-background-dark: #1F2631;
15 $c-background-darker: #1F232D;
16 $c-background-darkest--: #191D26;
17 $c-background-darkest: #13171D;
18 $c-accent: #B368E6;
19 $c-favorite: #F34C56;
20 $c-danger-background: #e7c2c2;
21 $c-danger-foreground: #6f2323;
22 $c-success-background: #9ad6a8;
23 $c-success-foreground: #0d4419;
```

```
1 /**
2  * @filesource _typography.scss
3  * @brief Fichier contenant toutes les polices utilisées dans
4  * mon application
5  * @author Tanguy Cavagna <tanguy.cvgn@eduge.ch>
6  * @date 2020-04-18
7  * @version 1.0.0
8 */
9 @font-face {
10    font-family: 'Regular';
11    src: url('../fonts/Poppins-Regular.ttf');
12 }
13
14 @font-face {
15    font-family: 'Nav link';
16    src: url('../fonts/Poppins-Bold.ttf');
17 }
18
19 @font-face {
20    font-family: 'Section title';
21    src: url('../fonts/Poppins-ExtraBold.ttf');
22 }
23
24 $f-regular: 'Regular';
25 $f-nav-link: 'Nav link';
26 $f-section-title: 'Section title';
```

```
1 """Contient le modèle d'une liste
2
3 @author: Tanguy Cavagna
4 @copyright: Copyright 2020, TPI
5 @version: 1.0.0
6 @date: 2020-05-28
7 """
8
9 class List:
10     """Modèle d'une liste
11     """
12
13     def __init__(self, list_id: int, name: str):
14         self.id = list_id
15         self.name = name
16
17     def serialize(self) -> dict:
18         """Sérialise les données
19         """
20
21         return {
22             'id': self.id,
23             'name': self.name
24         }
```

```
1 """Contient le modèle d'un type
2
3 @author: Tanguy Cavagna
4 @copyright: Copyright 2020, TPI
5 @version: 1.0.0
6 @date: 2020-05-26
7 """
8
9 class Type:
10     """Modèle d'un type
11     """
12
13     def __init__(self, type_id: int, name: str):
14         self.id = type_id
15         self.name = name
16
17     def serialize(self) -> dict:
18         """Sérialise les données
19         """
20
21         return {
22             'id': self.id,
23             'name': self.name
24         }
```

```
1 """Contient le modèle d'un utilisateur
2
3 @author: Tanguy Cavagna
4 @copyright: Copyright 2020, TPI
5 @version: 1.0.0
6 @date: 2020-05-27
7 """
8 from flask_login import UserMixin
9
10 class User(UserMixin):
11     """Modèle d'un utilisateur. Classe héritée de UserMixin
12     pour pouvoir être prise en compte comme classe de flask-login
13     """
14     def __init__(self, user_id: int, email: str, nickname: str):
15         self.id = user_id
16         self.email = email
17         self.nickname = nickname
18
19     def serialize(self) -> dict:
20         """Sérialise les données
21         """
22         return {
23             'id': self.id,
24             'email': self.email,
25             'nickname': self.nickname
26         }
```

```
1 """Contient le modèle d'un anime
2
3 @author: Tanguy Cavagna
4 @copyright: Copyright 2020, TPI
5 @version: 1.0.0
6 @date: 2020-05-28
7 """
8
9 class Anime:
10     """Modèle d'un anime
11     """
12
13     def __init__(self, anime_id: int, title: str, anime_type: str, episodes: int, status: str, picture: str, thumbnail: str, synonyms: str):
14         self.id = anime_id
15         self.title = title
16         self.type = anime_type
17         self.episodes = episodes
18         self.status = status
19         self.picture = picture
20         self.thumbnail = thumbnail
21         self.synonyms = synonyms
22         self.relations = None
23         self.is_favorite = None
24         self.in_list = None
25
26     def serialize(self) -> dict:
27         """Sérialise les données
28         """
29
30         return {
31             'id': self.id,
32             'title': self.title,
33             'type': self.type,
34             'episodes': self.episodes,
35             'status': self.status,
36             'picture': self.picture,
37             'thumbnail': self.thumbnail,
38             'synonyms': self.synonyms,
39             'relations': self.relations,
40             'is_favorite': self.is_favorite
41         }
```

```
1 """Contient le modèle d'un status
2
3 @author: Tanguy Cavagna
4 @copyright: Copyright 2020, TPI
5 @version: 1.0.0
6 @date: 2020-05-26
7 """
8
9 class Status:
10     """Modèle d'un statut
11     """
12
13     def __init__(self, status_id: int, name: str):
14         self.id = status_id
15         self.name = name
16
17     def serialize(self) -> dict:
18         """Sérialise les données
19         """
20
21         return {
22             'id': self.id,
23             'name': self.name
24         }
```

```
1 """Contient tout les loggers
2
3 @author: Tanguy Cavagna
4 @copyright: Copyright 2020, TPI
5 @version: 1.0
6 @date: 2020-05-26
7 """
8 import os
9
10 def log(e, in_terminal: bool = True, in_file: bool = True) ->
11     None:
12     """Log les erreurs
13
14     Arguments:
15         in_terminal {bool} -- Ajoute le log dans le
16         terminal
17         in_file {bool} -- Aout le log dans un fichier
18         """
19
20     try:
21         assert in_terminal or in_file, 'Au moins `in_terminal`'
22             ' ou `in_file` doit être à `True`'
23         assert e is not None, 'L\'exception donnée n\'est pas'
24             ' valide'
25
26         if in_terminal:
27             print(str(e))
28
29         if in_file:
30             # Écrit dans un fichier à la racine du projet
31             log_file_path = os.path.dirname(os.path.realpath(
32                 __file__)) + '/../log.txt'
33
34             append_write = 'a+' if os.path.exists(
35                 log_file_path) else 'w+'
36
37             with open(log_file_path, append_write) as log_file
38             :
39                 log_file.write(str(e) + '\n')
40                 log_file.close()
41
42     except AssertionError as assert_error:
43         log(assert_error)
44         raise
```

```

1  """Contient le contrôleur d'authentification
2
3 @author: Tanguy Cavagna
4 @copyright: Copyright 2020, TPI
5 @version: 1.0.0
6 @date: 2020-05-27
7 """
8 import re
9 from typing import Any
10
11 class BaseAuthenticationControlleur:
12     """Contrôleur pour l'authentification
13     """
14
15     def __init__(self, form):
16         """
17             Arguments:
18                 form {dict} -- Dictionnaire provenant de
19                 request.form
20         """
21         self.form = form
22         self._auth_errors = dict()
23
24     def update_auth_errors_dict(self, field_name: str,
25         error_condition: bool, message: str) -> bool:
26         """
27             Met à jour le dictionnaire d'erreurs si la
28             condition d'erreur passe
29
30             Arguments:
31                 field_name {str} -- Nom du champs pour lequel
32                 la condition s'applique
33                 error_condition {bool} -- Condition d'erreur.
34                 Si True, alors l'erreur doit être prise en compte.
35                 message {str} -- Message à afficher si l'
36                 erreur est levée
37
38             Returns:
39                 bool -- Est-ce que l'erreur a été levée
40         """
41
42         if error_condition:
43             if field_name not in self._auth_errors.keys():
44                 self._auth_errors.update({ field_name: [] })
45
46             self._auth_errors.get(field_name).append(message)
47
48         return self._auth_errors
49
50
51

```

```

42     def reset_auth_errors(self):
43         """Supprime toutes les anciennes erreurs
44         """
45         self._auth_errors = dict()
46
47     def get_state(self) -> Any:
48         """Est-ce que l'authentification est validé
49
50             Returns:
51                 bool -- `True` si aucune erreurs levées
52                 dict -- Toutes les erreurs levées
53         """
54
55     return self._auth_errors if self._auth_errors else
56     True
57
58     def field_empty(self, field_name: str, message: str = None
59 ) -> bool:
60         """Vérifie si le champ est vide
61
62             Arguments:
63                 field_name {str} -- Nom du champ à vérifier
64                 message {str} -- Message à afficher en cas d'
65                 erreur
66         """
67
68     error_condition = bool(self.form.get(field_name) == '')
69
70     if error_condition:
71         if message is None:
72             message = 'Ce champ est obligatoire'
73
74     return self.update_auth_errors_dict(field_name,
75     error_condition, message)
76
77     return None
78
79     # pylint: disable=redefined-builtin
80     def length(self, field_name: str, min: int = -1, max: int
81 = -1, message: str = None) -> bool:
82         """Vérifie si le champ respecte les longueurs imposée
83
84             Arguments:
85                 field_name {str} -- Nom du champ à vérifier
86                 min {int} -- Valeur minimum
87                 max {int} -- Valeur maximum
88                 message {str} -- Message à afficher en cas d'
89                 erreur

```

```

82     """
83         length = len(self.form.get(field_name))
84         error_condition = bool(length < min or (max != -1 and
85         length > max))
86
87         if error_condition:
88             if message is None:
89                 if max == -1: # Seulement une limite minimum
90                     est présent
91                     message = f'Le champ doit au moins avoir
92                     {min} caractères.'
93                 elif min == -1: # Seulement une limite
94                     maximum est présent
95                     message = f'Le champ ne peut pas excéder
96                     {max} caractères.'
97                 else:
98                     message = f'Le champ doit être compris
99                     entre {min} et {max} caractères'
100
101             return self.update_auth_errors_dict(field_name,
102             error_condition, message)
103
104         return None
105
106     def regex(self, field_name: str, regex: str, message: str
107     = None) -> bool:
108         """Vérifie si le champ valide le regex
109
110         Arguments:
111             field_name {str} -- Nom du champ à vérifier
112             regex {str} -- Expression régulière à
113             vérifier
114             message {str} -- Message à afficher en cas d'
115             erreur
116             """
117
118         p = re.compile(regex)
119         error_condition = bool(not p.match(self.form.get(
120             field_name)))
121
122         if error_condition:
123             if message is None:
124                 message = 'Champ invalide'
125
126         return self.update_auth_errors_dict(field_name,
127             error_condition, message)
128
129         return None

```

```

117
118     def email(self, field_name: str, message: str = None) ->
119         bool:
120             """Vérifie si le champ valide le regex
121
122             Arguments:
123                 field_name {str} -- Nom du champ à vérifier
124                 message {str} -- Message à afficher en cas d'
125                 erreur
126             """
127             if message is None:
128                 message = 'Email invalide'
129             return self.regex(field_name, r'^[a-zA-Z0-9._
130 -]+@[^\.]+\..+$', message)
131
132     def equal_to(self, field_name: str, other_field_name: str
133 , message: str = None) -> bool:
134             """Vérifie si le champ valide le regex
135
136             Arguments:
137                 field_name {str} -- Nom du champ à vérifier
138                 other_field_name {str} -- Nom du champ contre
139                 lequel tester l'égalité
140                 message {str} -- Message à afficher en cas d'
141                 erreur
142             """
143             try:
144                 field_to_equal = self.form.get(other_field_name)
145
146                 if field_to_equal is None:
147                     raise KeyError
148             except KeyError:
149                 self.update_auth_errors_dict(other_field_name,
150                 True, f'Nom de champs à égalé incorrect `{other_field_name}`')
151
152                 error_condition = bool(self.form.get(field_name) !=
153 field_to_equal)
154
155                 if error_condition:
156                     if message is None:
157                         message = f'Le champ doit être égale à {
158 other_field_name}'
159
160                 return self.update_auth_errors_dict(field_name,
161 error_condition, message)
162

```

```
153     return None
154
155 class SignupController(BaseAuthenticationControlleur):
156     """Contrôleur de l'inscription
157     """
158
159     def validate(self):
160         """Valide les champs du formulaire
161         """
162         if not self.field_empty('email'):
163             self.length('email', min=6, message='Choisissez
164             un email plus long')
165             self.email('email', 'Entrez un email valide')
166
167             self.field_empty('nickname')
168
169             if not self.field_empty('password'):
170                 self.length('password', min=6, message='
171                 Choisissez un mot de passe plus long')
172
173             if not self.field_empty('confirm'):
174                 self.equal_to('confirm', 'password', 'Les mots de
175                 passes doivent correspondre')
176
177             return not self._auth_errors
178
179
180     def validate(self):
181         """Valide les champs du formulaire
182         """
183         if not self.field_empty('email'):
184             self.email('email', 'Entrez un email valide')
185
186             self.field_empty('password')
187
188             return not self._auth_errors
189
```

```

1  """Contient la classe de contrôle des utilisateurs
2
3 @author: Tanguy Cavagna
4 @copyright: Copyright 2020, TPI
5 @version: 1.0.0
6 @date: 2020-05-28
7 """
8 from datetime import datetime as dt
9
10 from .SqliteController import SqliteError, SqliteController
11 from .logger import log
12 from ..models.List import List
13
14 class ListController:
15     """Controlleur d'une list
16     """
17
18     @classmethod
19     def __encapsulate_lists(cls, raw_results: dict) -> [List]:
20         """Encapsule les résultats d'une requete
21
22             Arguments:
23                 raw_resutls {dict} -- Résultats provenant de `SqliteController.execute`
24
25             Returns:
26                 [Anime] -- Liste de listes encapsuler
27             """
28
29     return [
30         List(result['idList'],
31               result['nameList'])
32         ) for result in raw_results
33     ]
34
35     @classmethod
36     def __serialize_encapsulated_lists(cls, encapsulated_lists
37 : list) -> [dict]:
38         """Sérialize toutes les listes contenu dans une liste
39             de liste encapsulé
40
41             Arguments:
42                 encapsulated_lists {list} -- Liste des animes
43                 encapsulé
44
45             Returns:
46                 [dict] -- Liste de tout les animes sérialisé
47             """

```

```

44         return [_list.serialize() for _list in
45             encapsulated_lists]
46     @classmethod
47     def get_defaults_for_user(cls, user_id: int) -> [List]:
48         """Retourne les listes par défaut d'un utilisateur
49
50             Arguments:
51                 user_id {int} -- Id de l'utilisateur en
52                 question
53
54             Returns:
55                 [List] -- Liste des listes par défaut de l'
56                 utilisateur
57         """
58         try:
59             sql_user_list = """SELECT list.idList, nameList
59                             FROM list
60                             JOIN user_has_list ON list.
61             idList = user_has_list.idList
62
63             WHERE user_has_list.idUser = ?
64             AND nameList IN ('Complétés', '
65             En cours', 'Abandonés', 'Planifiés')"""
66
67             results = SqliteController().execute(sql_user_list
68             , values=(user_id,), fetch_mode=SqliteController.FETCH_ALL)
69
70             encapsulated = cls.__encapsulate_lists(results)
71
72             return cls.__serialize_encapsulated_lists(
73                 encapsulated)
74         except SqliteError as e:
75             log(e)
76             return []
77
78     @classmethod
79     def get_list_of_an_anime(cls, anime_id: int) -> [List]:
80         """Récupère toutes les listes d'un anime
81
82             Arguments:
83                 anime_id {int} -- Id de l'anime
84
85             Returns:
86                 [List] -- Liste de toutes les listes de l'
87                 anime
88         """
89         try:
90
91
92

```

```

83             sqli_lists = """SELECT list.idList, nameList
84                 FROM list
85                 JOIN list_has_anime ON list.
86     idList = list_has_anime.idList
87                     WHERE list_has_anime.idAnime
88     = ?"""
89
90             results = SqliteController().execute(sqli_lists,
91     values=(anime_id,), fetch_mode=SqliteController.FETCH_ALL)
92
93             encapsulated = cls.__encapsulate_lists(results)
94
95             return cls.__serialize_encapsulated_lists(
96     encapsulated)
97         except SqliteError as e:
98             log(e)
99             return []
100
101     @classmethod
102     def get_user_lists(cls, user_id: int) -> [List]:
103         """Récupère toutes les listes d'un utilisateur
104
105         Arguments:
106             user_id {int} -- Id de l'utilisateur
107
108         Returns:
109             [List] -- Toutes les listes de l'utilisateur
110         """
111     try:
112         sql_user_lists = """SELECT list.idList, nameList
113                 FROM list
114                 JOIN user_has_list ON list.
115     idList = user_has_list.idList
116                     WHERE user_has_list.idUser
117     = ?"""
118
119             results = SqliteController().execute(
120     sql_user_lists, values=(user_id,), fetch_mode=
121     SqliteController.FETCH_ALL)
122
123             encapsulated = cls.__encapsulate_lists(results)
124
125             return cls.__serialize_encapsulated_lists(
126     encapsulated)
127         except SqliteError as e:
128             log(e)
129             return []

```

```

121
122     @classmethod
123     def get_by_id(cls, list_id: int) -> List:
124         """Récupère une liste via son id
125
126             Arguments:
127                 list_id {int} -- Id de la liste
128
129             Returns:
130                 List -- Liste trouvée
131
132         """
133         try:
134             sql_select = "SELECT idList, nameList FROM list
135 WHERE idList = ?"
136
137             row = SqliteController().execute(sql_select,
138                                         values=(list_id,), fetch_mode=SqliteController.FETCH_ONE)
139
140             return List(row['idList'], row['nameList'])
141         except SqliteError as e:
142             log(e)
143             return None
144
145     @classmethod
146     def add_new_list(cls, new_name: str, user_id: int) ->
147         List:
148             """Ajoute une nouvelle liste pour l'utilisateur
149             connecté
150
151             Arguments:
152                 new_name {str} -- Nom de la nouvelle liste
153                 user_id {int} -- Id de l'utilisateur connecté
154
155             Returns:
156                 List -- Nouvelle liste créée
157
158         """
159         try:
160             sql_insert = "INSERT INTO list(nameList,
161                                         modificationDate) VALUES(?, ?)"
162             sql_link_to_user = "INSERT INTO user_has_list(
163                                        idUser, idList, modificationDate) VALUES(?, ?, ?)"
164
165             current_date = dt.now().strftime('%Y-%m-%d %H:%M
166                                         :%S')
167
168             list_id = SqliteController().execute(sql_insert,
169                                         values=(new_name, current_date,), fetch_mode=SqliteController

```

```

159 .NO_FETCH)
160         SqliteController().execute(sql_link_to_user,
161         values=(user_id, list_id, current_date,), fetch_mode=
162         SqliteController.NO_FETCH)
163     except SqliteError as e:
164         log(e)
165     return None
166
167     @classmethod
168     def delete_by_id(cls, list_id: int, user_id: int) -> bool
169     :
170         """Supprime une liste via son id
171
172         Arguments:
173             list_id {int} -- Id de la list à supprimer
174             user_id {int} -- Id de l'utilisateur connecté
175
176         Returns:
177             bool -- États de la requete
178
179         try:
180             sql_unlink_anime = "DELETE FROM list_has_anime
181             WHERE idList = ?"
182             sql_unlink_to_user = "DELETE FROM user_has_list
183             WHEREidUser = ? AND idList = ?"
184             sql_delete = "DELETE FROM list WHERE idList = ?"
185
186             SqliteController().execute(sql_unlink_anime,
187             values=(list_id,), fetch_mode=SqliteController.NO_FETCH)
188             SqliteController().execute(sql_unlink_to_user,
189             values=(user_id, list_id,), fetch_mode=SqliteController.
190             NO_FETCH)
191             SqliteController().execute(sql_delete, values=(
192             list_id,), fetch_mode=SqliteController.NO_FETCH)
193
194             return True
195         except SqliteError as e:
196             print(str(e))
197             raise e
198
199     @classmethod
200     def rename(cls, list_id: int, new_list_name: str) -> bool
201     :
202         """Renomme une liste
203

```

```
196             Arguments:  
197                 list_id {int} -- Id de la liste à renommé  
198                 new_list_name {str} -- Nouveau nom de la  
     liste  
199  
200             Returns:  
201                 bool -- État de la requête  
202             """  
203         try:  
204             sql_update = "UPDATE list SET nameList = ? WHERE  
     idList = ?"  
205  
206             SqliteController().execute(sql_update, values=(  
     new_list_name, list_id,), fetch_mode=SqliteController.  
     NO_FETCH)  
207  
208             return True  
209         except SqliteError as e:  
210             log(e)  
211             return False  
212
```

```

1  """Contient la classe de contrôle des types
2  Les types d'anime sont : Special, Movie, ONA, TV
3
4  @author: Tanguy Cavagna
5  @copyright: Copyright 2020, TPI
6  @version: 1.0.0
7  @date: 2020-05-26
8  """
9  from sqlite3 import Error as SqliteError
10
11 from .SqliteController import SqliteController
12 from ..models.Type import Type
13 from .logger import log
14
15 class TypeController:
16     """Contrôleur des types
17     """
18
19     def __init__(self):
20         pass
21
22     @classmethod
23     def get_all(cls) -> [Type]:
24         """Récupère tout les types
25
26             Returns:
27                 [Type] -- Listes contenant des types
28         """
29
30         try:
31             rows = SqliteController().execute("SELECT `idType`",
32                                              `, `nameType` FROM `type`", fetch_mode=SqliteController().  
33             FETCH_ALL)
34             types = [Type(t['idType'], t['nameType']) for t in
35             rows]
36
37             return types
38         except SqliteError as e:
39             log(e)
40             raise e
41
42     @classmethod
43     def get_all_as_dict(cls) -> dict:
44         """Récupère tout les types sous forme de dictionnaire
45
46             Returns:
47                 dict -- Dictionnaire contenant les types avec
48                 le format "name": id

```

```
44      """
45      types = {}
46      for t in cls.get_all():
47          types.update({t.name: t.id})
48
49      return types
```

```
1  """Contient la classe de contrôle des utilisateurs
2
3 @author: Tanguy Cavagna
4 @copyright: Copyright 2020, TPI
5 @version: 1.0.0
6 @date: 2020-05-27
7 """
8 from datetime import datetime as dt
9
10 from sqlite3 import Error as SqliteError
11 from .SqliteController import SqliteController
12 from .TypeController import TypeController
13 from ..models.User import User
14 from .logger import log
15
16 class UserController:
17     """Contrôleur d'un utilisateur
18     """
19
20     @classmethod
21     def exists(cls, email: str) -> bool:
22         """Test si l'email existe déjà en base
23
24             Arguments:
25                 email {str} -- Email à tester
26
27             Returns:
28                 bool -- Est-ce que l'email existe
29         """
30
31         try:
32             sql_exists = "SELECT COUNT(*) AS `Count` FROM user
33 WHERE emailUser = ?"
34
35             count = SqliteController().execute(sql_exists,
36 values=(email,), fetch_mode=SqliteController.FETCH_ONE)[['Count']]
37
38             return bool(count > 0)
39         except SqliteError as e:
40             log(e)
41             return True
42
43     @classmethod
44     def check_password(cls, email: str, hashed_password: str
45 ) -> bool:
46         """Test la validité du mot de passe
```

```

44             Arguments:
45                 email {str} -- Email de l'utilisateur à tester
46                 hashed_password {str} -- Mot de passe à tester
47
48             Returns:
49                 bool -- Est-ce que le mot de passe est valide
50             """
51
52     try:
53         sql_check = "SELECT COUNT(*) AS `Count` FROM user
54 WHERE emailUser = ? AND password = ?"
55
56         count = SqliteController().execute(sql_check,
57 values=(email, hashed_password,), fetch_mode=SqliteController.
58 FETCH_ONE)[ 'Count' ]
59
60         return bool(count == 1)
61     except SqliteError as e:
62         log(e)
63         return False
64
65     @classmethod
66     def insert(cls, email: str, nickname: str, password: str
67 ) -> User:
68         """Insère un nouvel utilisateur en base
69
70             Arguments:
71                 email {str} -- Email
72                 nickname {str} -- Pseudo
73                 passowrd {str} -- Mot de passe
74
75             Returns:
76                 User -- Nouvel utilisateur créé
77             """
78
79     try:
80         sql_insert = "INSERT INTO user(emailUser, password
81 , nicknameUser, modificationDate) VALUES(?, ?, ?, ?)"
82
83         values_user = (
84             email,
85             password,
86             nickname,
87             dt.now().strftime('%Y-%m-%d %H:%M:%S')
88         )
89
90         last_row_id = SqliteController().execute(
91             sql_insert, values=values_user, fetch_mode=SqliteController.
92 NO_FETCH)

```

```

84
85             return User(last_row_id, email, nickname)
86     except SqliteError as e:
87         log(e)
88         return None
89
90     @classmethod
91     def get_by_id(cls, user_id: int) -> User:
92         """Récupère l'utilisateur via son id
93
94         Arguments:
95             user_id {int} -- Id de l'utilisateur à
96             récupérer
97
98         Returns:
99             User -- Utilisateur trouvé
100        """
101
102        try:
103            sql_select = "SELECT * FROM user WHERE idUser
104            = ?"
105
106            user_dict = SqliteController().execute(sql_select
107            , values=(user_id,), fetch_mode=SqliteController.FETCH_ONE)
108
109            if user_dict is not None:
110                return User(user_dict['idUser'], user_dict['
111 emailUser'], user_dict['nicknameUser'])
112
113            return None
114        except SqliteError as e:
115            log(e)
116            return None
117
118    @classmethod
119    def get_by_email(cls, email: str) -> User:
120        """Récupère un utilisateur via son email
121
122        Argument:
123            email {str} -- Email de l'utilisateur
124
125        Returns:
126            User -- Utilisateur trouvé
127        """
128
129        try:
130            sql_select = "SELECT * FROM user WHERE emailUser
131            = ?"
132
133            user_dict = SqliteController().execute(sql_select
134            , values=(email,), fetch_mode=SqliteController.FETCH_ONE)
135
136            if user_dict is not None:
137                return User(user_dict['idUser'], user_dict['
138 emailUser'], user_dict['nicknameUser'])
139
140            return None
141        except SqliteError as e:
142            log(e)
143            return None

```

```

126             user_dict = SqliteController().execute(sql_select
127             , values=(email,), fetch_mode=SqliteController.FETCH_ONE)
128
129             if user_dict is not None:
130                 return User(user_dict['idUser'], user_dict['
131 emailUser'], user_dict['nicknameUser']))
132
133             return None
134     except SqliteError as e:
135         log(e)
136         return None
137
138     @classmethod
139     def get_by_nickname(cls, nickname: str) -> User:
140         """Récupère un utilisateur via son pseudo
141
142         Arguments:
143             nickname {str} -- Pseudo de l'utilisateur
144             cherché
145
146             Returns:
147             User -- Utilisateur trouvé
148             """
149
150             try:
151                 sql_select = "SELECT * FROM user WHERE
152 nicknameUser = ?"
153
154                 user_dict = SqliteController().execute(sql_select
155             , values=(nickname,), fetch_mode=SqliteController.FETCH_ONE)
156
157                 if user_dict is not None:
158                     return User(user_dict['idUser'], user_dict['
159 emailUser'], user_dict['nicknameUser']))
160
161                     return None
162     except SqliteError as e:
163         log(e)
164         return None
165
166     @classmethod
167     def setup_default_lists(cls, user_id: int) -> bool:
168         """Créer les listes par défaut si inexistantes
169
170         Arguments:
171             user_id {int} -- Id de l'utilisateur
172
173         Returns:

```

```

167             bool -- `True` si tout c'est bien passé
168         """
169     try:
170         sql_default_list = "INSERT INTO list(nameList,
171                                         modificationDate) VALUES(?, ?)"
172         sql_link_list_user = "INSERT INTO user_has_list(
173                                         idUser, idList, modificationDate) VALUES(?, ?, ?)"
174
175         values_list = [
176             'Complétés',
177             'En cours',
178             'Abandonés',
179             'Planifiés'
180         ]
181
182         current_time = dt.now().strftime('%Y-%m-%d %H:%M
183 :%S')
184         for list_name in values_list:
185             list_id = SqliteController().execute(
186                 sql_default_list, values=(list_name, current_time,), fetch_mode=SqliteController.NO_FETCH)
187             SqliteController().execute(sql_link_list_user
188             , values=(user_id, list_id, current_time,), fetch_mode=
189             SqliteController.NO_FETCH)
190
191             return True
192     except SqliteError as e:
193         log(e)
194         return False
195
196     @classmethod
197     def get_stats_by_id(cls, user_id: int) -> []:
198         """Récupère les statistiques pour un utilisateur
199
200         Arguments:
201             user_id {int} -- Id de l'utilisateur
202
203         Returns:
204             [] -- Liste des statistiques
205         """
206
207     try:
208         sql_stats = """SELECT COUNT(*) AS `Count`
209                     FROM anime
210                     WHERE idAnime IN (
211                         SELECT list_has_anime.idAnime
212                         FROM user_has_list
213                         JOIN list_has_anime ON
214                             list_has_anime.idList = user_has_list.idList
215                         AND list_has_anime.idUser = user_has_list.idUser
216                     )"""
217
218         stats = SqliteController().execute(sql_stats, values=(user_id,))
219
220         if stats.fetchone():
221             count = stats.fetchone()
222             return count['Count']
223
224     except SqliteError as e:
225         log(e)
226         return None
227
228     return []

```

```
206 user_has_list.idList = list_has_anime.idList
207     WHERE user_has_list.idUser = ?
208     AND user_has_list.idList = (
209         SELECT list.idList
210         FROM list
211         JOIN user_has_list ON list
212             WHERE user_has_list.idUser
213                 AND list.nameList LIKE "
214                     )
215                     )
216                     AND type = (
217                         SELECT idType
218                             FROM type
219                             WHERE nameType LIKE ?
220                         )"""
221
222     types = TypeController().get_all()
223
224     stats = {}
225     for t in types:
226         stats[t.name] = SqliteController().execute(
227             sql_stats, values=(user_id,user_id,t.name,), fetch_mode=
228             SqliteController.FETCH_ONE)['Count']
229
230     return stats
231 except SqliteError as e:
232     log(e)
233     return []
```

```

1  """Contient la classe de contrôle des animes
2
3 @author: Tanguy Cavagna
4 @copyright: Copyright 2020, TPI
5 @version: 1.0
6 @date: 2020-05-26
7 """
8 import json
9 import pathlib
10 #import sys
11 #import os
12 from sqlite3 import Error as SqliteError
13 from datetime import datetime as dt
14 from random import randint
15
16 from .SqliteController import SqliteController
17 from .TypeController import TypeController
18 from .StatusController import StatusController
19 from .ListController import ListController
20 from ..models.Anime import Anime
21 from .logger import log
22
23 class AnimeController:
24     """Controlleur d'un anime
25 """
26
27     #pylint: disable=too-many-return-statements
28     @classmethod
29     def override_local_with_json(cls, path: str) -> None:
30         """Écrase toutes les données en base
31
32             Arguments:
33                 path {string} -- Chemin du fichier json
34 """
35         with open(pathlib.Path(__file__).parent / path, 'r',
36 encoding="utf8") as f:
37             animes = json.load(f)
38
39     try:
40         # Mise en place des tables si elles ne sont pas
41         # présente et truncate tout ce qui est en lien direct avec la
42         # table 'anime'
43         if not SqliteController().setup_type_table():
44             return False
45         if not SqliteController().setup_status_table():
46             return False
47         if not SqliteController().setup_anime_table():
48

```

```

42     return False
43         if not SqliteController().setup_url_table():
44             return False
45             if not SqliteController().setup_user_table():
46                 return False
47                 if not SqliteController().setup_list_table():
48                     return False
49                     if not SqliteController().setup_user_has_list_table(): return False
50                     if not SqliteController().setup_list_has_anime_table(): return False
51                     if not SqliteController().setup_user_has_favorite_table(): return False
52                     if not SqliteController().truncate_anime_related():
53                         return False
54
55             # Requêtes pour les insertions de nouvelles
56             données
57             sql_query_type = "INSERT INTO type(nameType,
58               modificationDate) VALUES(?, ?)"
59             sql_query_status = "INSERT INTO status(nameStatus
60               , modificationDate) VALUES(?, ?)"
61             sql_query_anime = "INSERT INTO anime(idAnime,
62               title, type, episodes, status, picture, thumbnail, synonyms,
63               modificationDate) VALUES(?, ?, ?, ?, ?, ?, ?, ?, ?, ?)"
64             sql_query_anime_ft = "INSERT INTO anime_ft(idAnime
65               , title) VALUES(?, ?)"
66             sql_query_url = "INSERT INTO url(urlName, idAnime
67               , isRelation, modificationDate) VALUES(?, ?, ?, ?)"
68
69             # Tableaux ayant pour but de stocké les futures
70             valeurs à mettre dans les table pour faire un insert_many
71             values_type = []
72             values_status = []
73             values_anime = []
74             values_anime_ft = []
75             values_source = []
76             values_relation = []
77
78             current_time = dt.now().strftime('%Y-%m-%d %H:%M:%
79             S')
80
81             yield "Récupération des types et statuts"
82
83             # Récupération des statuts et des types
84             for anime in animes['data']:
85                 # Suppression de tout les OVA car souvent

```

```
72 ayant un contenu innapproprié pour un TPI
73             if anime['type'] == 'OVA':
74                 continue
75
76             # Ajout distinct des types
77             if not any(anime['type'] in i for i in
78                 values_type):
79                 values_type.append((
80                     anime['type'],
81                     current_time
82                 ))
83
84             # Ajout distinct des statuts
85             if not any(anime['status'] in i for i in
86                 values_status):
87                 values_status.append((
88                     anime['status'],
89                     current_time
90
91         yield "Insertion des types et statuts"
92         SqliteController().execute_many(sql_query_type,
93             values_type)
94         SqliteController().execute_many(sql_query_status
95             , values_status)
96
97
98         # Reparcourir le fichier json pour extraire cette
99         # fois si les animes et leurs relations
100        idAnime = 1
101        for anime in animes['data']:
102            # Suppression de tout les OVA car leur
103            # contenu est souvent innapproprié pour un TPI
104            if anime['type'] == 'OVA':
105                continue
106
107            values_anime.append((
108                idAnime,
109                anime['title'],
110                types[anime['type']],
111                anime['episodes'],
112                status[anime['status']],
113                anime['picture'],
114                anime['thumbnail'],
```

```

113             ',' .join(anime[ 'synonyms' ]) ) if len(anime[  

114             'synonyms' ]) > 0 else None,  

115                     current_time  

116             ))  

117             values_anime_ft.append(  

118                 idAnime,  

119                 anime[ 'title' ]  

120             ))  

121  

122         for source in anime[ 'sources' ]:  

123             values_source.append((  

124                 source,  

125                 idAnime,  

126                 False,  

127                 current_time  

128             ))  

129  

130         for relation in anime[ 'relations' ]:  

131             values_relation.append((  

132                 relation,  

133                 idAnime,  

134                 True,  

135                 current_time  

136             ))  

137  

138         idAnime += 1  

139  

140         yield "Insertion des animes et urls"  

141             SqliteController().execute_many(sql_query_anime,  

142             values_anime)  

143             SqliteController().execute_many(  

144             sql_query_anime_ft, values_anime_ft)  

145             SqliteController().execute_many(sql_query_url,  

146             values_source)  

147             SqliteController().execute_many(sql_query_url,  

148             values_relation)  

149  

150         yield "redirect"  

151         return True  

152     except SqliteError as e:  

153         log(e)  

154         return False  

155  

156     @classmethod  

157     def __encapsulate_animes(cls, raw_results: dict) -> [  

158         Anime]:

```

```

154         """Encapsule les résultats d'une requête
155
156             Arguments:
157                 raw_results {dict} -- Résultats provenant de
`SqliteController.execute`
158
159             Returns:
160                 [Anime] -- Liste d'anime encapsuler
161             """
162     return [
163         Anime(result['idAnime'],
164               result['title'],
165               result['nameType'],
166               result['episodes'],
167               result['nameStatus'],
168               result['picture'],
169               result['thumbnail'],
170               result['synonyms'])
171     ) for result in raw_results
172   ]
173
174 @classmethod
175 def __serialize_encapsulated_animes(cls,
encapsulated_animes: list) -> [dict]:
176     """Sérialise tout les animes contenu dans une liste d
'anime encapsulé
177
178             Arguments:
179                 encapsulated_animes {list} -- Liste des
animes encapsulé
180
181             Returns:
182                 [dict] -- Liste de tout les animes sérialisé
183             """
184     return [anime.serialize() for anime in
encapsulated_animes]
185
186 @classmethod
187 def __get_count(cls) -> int:
188     """Retourne le nombre d'anime présent en base
189
190             Returns:
191                 int -- Nombre d'anime en base
192             """
193     try:
194         sql_count = """SELECT COUNT(*) AS `Count` FROM
anime"""

```

```

195
196         count = SqliteController().execute(sql_count,
197         fetch_mode=SqliteController.FETCH_ONE)['Count']
198
199     return count
200 except SqliteError as e:
201     log(e)
202     return None
203
204     @classmethod
205     def get_from_search_string(cls, search_string: str) -> [Anime]:
206         """Retourne les animes concordant avec la chaine de recherche
207
208         Arguments:
209             search_string {str} -- Chaine de recherche
210
211         Returns:
212             [Anime] -- Liste des animes concordant
213
214         """
215         try:
216             # Le join sur la table virtuelle est obligatoire
217             # pour réalisé des recherches fulltext
218             sql_search = """SELECT anime.idAnime, anime.title,
219             , nameType, episodes, nameStatus, picture, thumbnail,
220             synonyms
221
222             FROM anime
223             JOIN anime_ft ON anime.idAnime =
224             anime_ft.idAnime
225             JOIN status ON anime.status =
226             JOIN type ON anime.type = type.
227             idType
228             WHERE anime_ft.title MATCH ?
229             LIMIT 9"""
230
231
232             results = SqliteController().execute(sql_search,
233             values=(search_string,), fetch_mode=SqliteController.
234             FETCH_ALL)
235
236             encapsulated = cls.__encapsulate_animes(results)
237
238             return cls.__serialize_encapsulated_animes(
239             encapsulated)
240 except SqliteError as e:
241     log(e)

```

```

230         return []
231
232     @classmethod
233     def get_relations_by_anime_id(cls, anime_id: int) -> [Anime]:
234         """Retourne toutes les relations d'un anime
235
236         Arguments:
237             anime_id {int} -- Id de l'anime duquel nous
238             voulons les relations
239
240         Returns:
241             [Anime] -- Liste de toutes les relations
242         """
243
244     try:
245         # Cette requête est relativement complexe. Pour
246         # avoir toutes les relations d'un anime, voici comment elle
247         # procède :
248         #   Récupérer toutes les url liées à un anime,
249         #   récupérer tout les ids des animes ayant des urls référencé
250         #   comme relation à l'anime principal
251         #   et ne pas reprendre une seconde fois l'anime
252         #   principale. Ne reste plus qu'à récupérer les informations de
253         #   ces animes
254         sql_relations = """SELECT anime.idAnime, anime.
255                         title, nameType, episodes, nameStatus, picture, thumbnail,
256                         synonyms
257                         FROM anime
258                         JOIN status ON anime.status =
259                         status.idStatus
260                         JOIN type ON anime.type = type
261                         .idType
262                         WHERE idAnime IN (
263                             SELECT idAnime
264                             FROM url
265                             WHERE urlName IN (
266                                 SELECT urlName
267                                 FROM url
268                                 WHERE idAnime = ?
269                             )
270                             AND isRelation = 1
271                             AND idAnime != ?
272                         )"""
273
274         results = SqliteController().execute(
275             sql_relations, values=(anime_id, anime_id,), fetch_mode=
276             SqliteController.FETCH_ALL)

```

```

263             encapsulated = cls.__encapsulate_animes(results)
264
265         return cls.__serialize_encapsulated_animes(
266             encapsulated)
267     except SqliteError as e:
268         log(e)
269         return []
270
271     @classmethod
272     def get_random_anime(cls) -> Anime:
273         """Récupère un anime aléatoire
274
275         Returns:
276             Anime -- Anime tiré aléatoirement
277         """
278
279         try:
280             sql_select = """SELECT anime.idAnime, anime.title
281             , nameType, episodes, nameStatus, picture, thumbnail,
282             synonyms
283             FROM anime
284             JOIN status ON anime.status =
285             status.idStatus
286             JOIN type ON anime.type = type.
287             idType
288             WHERE idAnime = ?"""
289
290             random_anime_id = randint(1, cls.__get_count())
291
292             results = SqliteController().execute(sql_select,
293             values=(random_anime_id,), fetch_mode=SqliteController.
294             FETCH_ONE)
295
296             encapsulated = cls.__encapsulate_animes([results
297             ])
298
299         return cls.__serialize_encapsulated_animes(
300             encapsulated)[0]
301     except SqliteError as e:
302         log(e)
303         return None
304
305     @classmethod
306     def is_anime_in_user_favorite(cls, user_id: int, anime_id:
307         int) -> bool:
308         """Retourne si l'anime est dans la liste des favoris
309         de l'utilisateur

```

```

299
300             Arguments:
301                 user_id {int} -- Id de l'utilisateur en
302                     question
303                     anime_id {int} -- Id de l'anime sur lequel
304                     tester si il est favoris
305
306             Returns:
307                 bool -- Est-ce que l'anime est dans les
308                     favoris de l'utilisateur
309                     """
310
311     try:
312         sql_is_favorite = """SELECT COUNT(*) AS `Count`  

313                             FROM user_has_favorite  

314                             WHERE idUser = ?  

315                             AND idAnime = ?"""
316
317         results = SqliteController().execute(
318             sql_is_favorite, values=(user_id,anime_id,), fetch_mode=
319             SqliteController.FETCH_ONE)['Count']
320
321         return bool(results == 1)
322     except SqliteError as e:
323         log(e)
324         return False
325
326     @classmethod
327     def is_anime_in_list(cls, anime_id: int, list_id: int
328 ) -> bool:
329         """Est-ce que l'anime est dans la liste
330
331             Arguments:
332                 anime_id {int} -- Id de l'anime
333                 list_id {int} -- Id de la liste
334
335             Returns:
336                 bool -- Est-ce que l'anime est dans la liste
337                     """
338
339     try:
340         sql_is_in_list = """SELECT COUNT(*) AS `Count`  

341                             FROM list_has_anime  

342                             WHERE idAnime = ?  

343                             AND idList = ?"""
344
345         results = SqliteController().execute(
346             sql_is_in_list, values=(anime_id,list_id,), fetch_mode=
347             SqliteController.FETCH_ONE)['Count']

```

```

338
339         return bool(results == 1)
340     except SqliteError as e:
341         log(e)
342         return False
343
344     @classmethod
345     def set_anime_in_user_favorite(cls, user_id: int,
346                                    anime_id: int) -> bool:
347         """Met à jour l'état de favoris d'un anime pour un
348         utilisateur
349
350         Arguments:
351             user_isd {int} -- Id de l'utilisateur
352             anime_id {int} -- Id de l'anime
353
354         Returns:
355             bool -- État de favoris de l'anime
356         """
357
358         try:
359             sql_delete_favorite = """DELETE FROM
360             user_has_favorite
361             WHERE idAnime = ?
362             AND idUser = ?"""
363
364             sql_get_last_order_id = """SELECT orderId
365             FROM user_has_favorite
366             WHERE idUser = ?
367             ORDER BY orderId DESC
368             LIMIT 1"""
369
370             sql_add_favorite = "INSERT INTO user_has_favorite
371             (idAnime, idUser, orderId, modificationDate) VALUES
372             (?, ?, ?, ?)"
373
374             if cls.is_anime_in_user_favorite(user_id,
375                                            anime_id):
376                 SqliteController().execute(
377                     sql_delete_favorite, values=(anime_id, user_id, ), fetch_mode=
378                     SqliteController.NO_FETCH)
379
380                 # Réattribué les orderId
381
382                 is_favorite = False
383             else:
384                 last_order_id = SqliteController().execute(
385                     sql_get_last_order_id, values=(user_id, ), fetch_mode=
386                     SqliteController.FETCH_ONE)
387                 last_order_id = int(last_order_id['orderId'])

```

```

374 ]) + 1 if last_order_id is not None else 1
375                         SqliteController().execute(sql_add_favorite,
376                                         values=(anime_id,user_id,last_order_id,dt.now().strftime('%Y
377                                         -%m-%d %H:%M:%S')),), fetch_mode=SqliteController.NO_FETCH)
378
379             is_favorite = True
380
381     return is_favorite
382 except SqliteError as e:
383     log(e)
384     return False
385
386 @classmethod
387 def get_favorite_by_user_id(cls, user_id: int) -> [Anime
388 ]:
389     """Récupère tout les favoris d'un utilisateur
390
391     Arguments:
392         user_id {int} -- Id de l'utilisateur
393
394     Returns:
395         [Anime] -- Liste de tout les favoris
396     """
397
398     try:
399         sqlFavorites = """SELECT anime.idAnime, anime.
400 title, nameType, episodes, nameStatus, picture, thumbnail,
401 synonyms
402
403         FROM anime
404         JOIN status ON anime.status =
405             status.idStatus
406
407         JOIN type ON anime.type = type
408
409         JOIN user_has_favorite ON
410             anime.idAnime = user_has_favorite.idAnime
411
412             WHERE user_has_favorite.idUser
413
414             = ?
415
416             ORDER BY orderId ASC"""
417
418
419         results = SqliteController().execute(
420             sqlFavorites, values=(user_id,), fetch_mode=SqliteController
421             .FETCH_ALL)
422
423         encapsulated = cls.__encapsulate_animes(results)
424
425     return cls.__serialize_encapsulated_animes(
426         encapsulated)
427
428 except SqliteError as e:

```

```

409         log(e)
410         return []
411
412     @classmethod
413     def set_anime_in_list(cls, anime_id: int, list_id: int
414 ) -> bool:
415         """Met à jour l'appartenance d'un anime à une liste
416
417         Arguments:
418             anime_id {int} -- Id de l'anime
419             list_id {int} -- Id de la liste
420
421     try:
422         sql_delete_from_list = """DELETE FROM
423 list_has_anime
424
425             WHERE idAnime = ?
426             AND idList = ?"""
427
428         sql_add_to_list = """INSERT INTO list_has_anime(
429             idAnime, idList, modificationDate)
430
431             VALUES(?, ?, ?)"""
432
433
434         if cls.is_anime_in_list(anime_id, list_id):
435             SqliteController().execute(
436                 sql_delete_from_list, values=(anime_id, list_id,), fetch_mode=
437                 SqliteController.NO_FETCH)
438             is_in_list = False
439
440         else:
441             SqliteController().execute(sql_add_to_list,
442                 values=(anime_id, list_id, dt.now().strftime('%Y-%m-%d %H:%M:%S
443 ')), fetch_mode=SqliteController.NO_FETCH)
444             is_in_list = True
445
446
447         return is_in_list
448     except SqliteError as e:
449         log(e)
450         return False
451
452
453     @classmethod
454     def delete_anime_status(cls, user_id: int, anime_id: int
455 ) -> bool:
456         """Supprime le stats actuel d'un anime pour un
457         utilisateur
458
459         Arguments:
460             user_id {int} -- Id de l'utilisateur
461             anime_id {int} -- Id de l'anime
462
463
464
465
466

```

```

447         Returns:
448             bool -- État de la requête
449             """
450     try:
451         sql_delete = """DELETE FROM list_has_anime
452                         WHERE idList IN (?, ?, ?, ?, ?)
453                         AND idAnime = ?"""
454
455         default_lists = ListController().
456         get_defaults_for_user(user_id)
457         in_clause = [l['id'] for l in default_lists] # 
Créer un tableau regroupant les ids des listes par défaut
458         values = [] + in_clause # Fusionne les deux
listes
459         values.append(anime_id)
460
461         SqliteController().execute(sql_delete, values=
tuple(values), fetch_mode=SqliteController.NO_FETCH)
462
463         return True
464     except SqliteError as e:
465         log(e)
466         return False
467
468     @classmethod
469     def get_animes_in_list_for_user(cls, user_id: int,
list_id: int, search_terms: str = None) -> [Anime]:
470         """Récupère la liste de tout les animes d'une liste
471
Arguments:
472             user_id {int} -- Id de l'utilisateur
473             list_id {int} -- Id de la liste
474             search_terms {str} -- Chaine de caractères
pour la recherche d'anime dans une liste
475         Returns:
476             [Anime] -- Liste de tous les animes trouvés
477             """
478     try:
479         animes = {}
480
481         sql_select = """SELECT list.idList, list.nameList
, anime.idAnime, anime.title, nameType, list_has_anime.idList
, anime.episodes, nameStatus, anime.picture, anime.thumbnail
, anime.synonyms
482
FROM anime
483 JOIN anime_ft ON anime.idAnime =
anime_ft.idAnime

```

```

484                                     JOIN list_has_anime ON
485                                     list_has_anime.idAnime = anime.idAnime
486                                     JOIN user_has_list ON
487                                     user_has_list.idList = list_has_anime.idList
488                                     JOIN list ON list.idList =
489                                     list_has_anime.idList
490                                     JOIN type ON type.idType = anime.
491                                     type
492                                     JOIN status ON status.idStatus =
493                                     anime.status
494                                     WHERE user_has_list.idUser = ?"""
495
496                                     if search_terms is not None and search_terms != "":
497                                         sql_select += " AND anime_ft.title MATCH ?"
498
499                                         if list_id is not None:
500                                             sql_select += " AND list_has_anime.idList
501                                             = ?"
502                                             values = (user_id, search_terms, list_id
503                                             ,)
504                                         else:
505                                             values = (user_id, search_terms, )
506                                         elif list_id is not None:
507                                             sql_select += " AND list_has_anime.idList
508                                             = ?"
509                                             values = (user_id, list_id, )
510                                         else:
511                                             values = (user_id, )
512
513                                         sql_select += " ORDER BY list.idList"
514
515                                         rows = SqliteController().execute(sql_select,
516                                         values=values, fetch_mode=SqliteController.FETCH_ALL)
517
518                                         # Ajoute les animes dans la bonne liste
519                                         for row in rows:
520                                             new_name = row['nameList']
521                                             if new_name not in animes:
522                                                 animes[new_name] = []
523                                                 animes[new_name].append(Anime(row['idAnime']
524                                         ], row['title'], row['nameType'], row['episodes'], row[
525                                         'nameStatus'], row['picture'], row['thumbnail'], row[
526                                         'synonyms']).serialize())
527
528                                         return animes
529                                         except SqliteError as e:

```

```
518         log(e)
519         return {}
520
521     @classmethod
522     def reorderFavorites(cls, user_id: int, ids: list) ->
523         bool:
524             """Met à jour l'ordre des animes
525
526             Arguments:
527                 user_id {int} -- Id de l'utilisateur
528                 ids {list} -- Liste des ids des animes
529
530             Returns:
531                 bool -- État de la mise à jour
532             """
533
534     try:
535         sql_update = "UPDATE user_has_favorite SET
536         orderId = ?, modificationDate = ? WHERE idAnime = ? AND
537         idUser = ?"
538
539         order_id = 1
540         modification_date = dt.now().strftime('%Y-%m-%d %
541         H:%M:%S')
542
543         for anime_id in ids:
544             SqliteController().execute(sql_update, values
545             =(order_id, modification_date, anime_id, user_id,),,
546             fetch_mode=SqliteController.NO_FETCH)
547             order_id += 1
548
549     return True
550 except SqliteError as e:
551     log(e)
552     return False
553
```

```

1  """Contient la classe de contrôle de la base de données MySQL
2
3 @author: Tanguy Cavagna
4 @copyright: Copyright 2020, TPI
5 @version: 1.0.0
6 @date: 2020-06-04
7 """
8 from typing import Any
9
10 import mysql.connector
11 from mysql.connector import Error
12 from .logger import log
13
14 class MySQLController:
15     """Classe permettant d'accéder à ma base de données MySQL
16     """
17
18     FETCH_ALL = 0
19     FETCH_ONE = 1
20     NO_FETCH = 2
21
22     def __init__(self):
23         """Constructeur vide
24         """
25
26     def __getattr__(self, attr):
27         """Essaie d'accéder à une variable ou fonction
28         innexistante
29         """
30         return "Tried to handle unknown method / attribute : {0}".format(attr)
31
32     @classmethod
33     def __connection(cls) -> mysql.connector.MySQLConnection:
34         """Créer et retourne la connexion
35
36         Returns:
37             {mysql.connector.MySQLConnection} -- Nouvelle
38         connexion
39         """
40
41         return mysql.connector.connect(host='localhost',
42                                         database='tpi',
43                                         user='root',
44                                         password='root')
45
46     @classmethod
47     def openConnection(cls) -> mysql.connector.MySQLConnection

```

```

44 :
45     """Ouvre une connexion
46
47     Returns:
48         {mysql.connector.MySQLConnection} -- Connection
49         to the database
50     """
51
52     @classmethod
53     def closeConnection(cls, connection) -> None:
54         """Ferme une connexion données
55
56         Arguments:
57             connection {mysql.connector.MySQLConnection}
58         } -- Connexion à fermer
59     """
60
61     @classmethod
62     def executeMany(cls, connection, query, values) -> None:
63         """Fait une requete de type many. (INSERT INTO <table
64         > VALUES(...values), (...values), (...values), ...)
65
66         Arguments:
67             connection {mysql.connector.MySQLConnection}
68         } -- Connexion sur laquelle faire la requête
69             query {str} -- Requête à faire
70             values {tuple} -- Valeurs à attribué à la
71             requête
72     """
73
74     try:
75         cursor = connection.cursor(prepared=True)
76         cursor.executemany(query, values)
77         connection.commit()
78     except Error as e:
79         log(e)
80
81     @classmethod
82     def executePrepare(cls, connection, query, values = None,
83     fetch_mode = 0) -> Any:
84         """Éxecute une requête préparée
85
86         Arguments:
87             connection {mysql.connector.MySQLConnection}
88         } -- Connexion sur laquelle faire la requête
89             query {str} -- Requête à faire

```

```

84
85             Keyword Arguments:
86                 values {tuple} -- Valeurs à attribué à la
87                 requête (default: {None})
88                 fetch_mode {int} -- Type de fetch (default: {
89                 0})
90
91             Returns:
92                 Depending on which fetch mod you are, you
93                 could get bool or a list of tuples
94                 """
95
96             rows = None
97
98         try:
99             cursor = connection.cursor(prepared=True)
100
101             cursor.execute(query, values)
102
103             fields_names = [i[0] for i in cursor.description]
104
105             # Fetch datas
106             if fetch_mode == cls.FETCH_ALL:
107                 results = cursor.fetchall()
108
109                 rows = [dict(zip(fields_names, results[i]))]
110
111                 for i in range(len(results)):
112                     elif fetch_mode == cls.FETCH_ONE:
113                         rows = cursor.fetchone()
114                     elif fetch_mode == cls.NO_FETCH:
115                         rows = None
116
117                     # Return datas
118                     if rows is not None:
119                         return rows
120
121
122             return None
123         except Error as e:
124             log(e)
125             return False
126         finally:
127             cls.closeConnection(connection)
128
129     @classmethod
130     def executeQuery(cls, connection, query, values = None,
131     fetch_mode = 0, last_insert_id = False, close_connection =
132     True) -> Any:
133         """Execute and retrieve data from querie

```

```

125
126          Arguments:
127              connection {mysql.connector.MySQLConnection}
128      } -- Connexion sur laquelle faire la requête
129          query {str} -- Requête à faire
130
131          Keyword Arguments:
132              values {tuple} -- Valeurs à attribué à la
133                  requête (default: {None})
134              fetch_mode {int} -- Type de fetch (default: {
135                  0})
136                  last_insert_id {bool} -- Retourne le
137                      last_insert_id ou pas (default: {False})
138                  close_connection {bool} -- Fermer la
139                      connexion à la fin de l'exécution de la requête
140
141          Returns:
142              Depending on which fetch mod you are, you
143                  could get bool or a list of tuples
144
145          """
146          rows = None
147          last_id = None
148
149          try:
150              cursor = connection.cursor(dictionary=True)
151
152              # Execute the query
153              if (values is not None):
154                  cursor.execute(query, values)
155                  if not query.startswith("SELECT"):
156                      connection.commit()
157
158              else:
159                  cursor.execute(query)
160                  if query.startswith("DELETE"):
161                      connection.commit()
162
163
164                  # Fetch datas
165                  if fetch_mode == cls.FETCH_ALL:
166                      rows = cursor.fetchall()
167                  elif fetch_mode == cls.FETCH_ONE:
168                      rows = cursor.fetchone()
169                  elif fetch_mode == cls.NO_FETCH:
170                      rows = None
171
172
173                  if last_insert_id:
174                      last_id = cursor.lastrowid

```

```
166
167     if last_id is not None:
168         return last_id
169
170     # Return datas
171     if rows is not None:
172         return rows
173
174     return None
175 except Error as e:
176     log(e)
177     return False
178 finally:
179     if close_connection:
180         cls.closeConnection(connection)
181
```

```

1  """Contient la classe de contrôle de la base de données Sqlite
2
3 @author: Tanguy Cavagna
4 @copyright: Copyright 2020, TPI
5 @version: 1.0.0
6 @date: 2020-05-26
7 """
8 import sqlite3
9 from typing import Any
10 from sqlite3 import Error as SqliteError
11 from mysql.connector import Error as MysqlError
12 from flask import g
13
14 from .MySQLController import MySQLController
15 from .logger import log
16
17 class SqliteController:
18     """Classe permettant d'accéder à ma base de données Sqlite
19     """
20
21     FETCH_ALL = 0
22     FETCH_ONE = 1
23     NO_FETCH = 2
24
25     connection = None
26
27     @classmethod
28     def __get_cursor(cls) -> object:
29         """Récupère le curseur de ma connexion
30         """
31         return cls.get_instance().cursor()
32
33     #pylint: disable=no-self-use
34     @classmethod
35     def __dict_factory(cls, cursor: object, row: object) ->
36     dict:
37         """Retourne les valeurs du fetch comme dictionnaire
38             see: https://stackoverflow.com/questions/3300464/
39             how-can-i-get-dict-from-sqlite-query
40
41             Arguments:
42                 cursor {object} -- Cursor of the connection
43                 row {object} -- Returned rows
44
45             Returns:
46                 dict -- Full dict
47         """

```

```

46         d = {}
47         for idx, col in enumerate(cursor.description):
48             d[col[0]] = row[idx]
49         return d
50
51     @classmethod
52     def get_instance(cls) -> object:
53         """Récupère l'instance de connexion à la base
54         """
55         SqliteController.connection = getattr(g, '_database',
56         None)
56         if SqliteController.connection is None:
57             SqliteController.connection = g._database =
58             sqlite3.connect("/home/cavagnat/Documents/programmation/python
59             /Animanga/static/database/tpi.db", detect_types=sqlite3.
60             PARSE_DECLTYPES)
61             SqliteController.connection.row_factory = cls.
62             __dict_factory
63
64             return SqliteController.connection
65
66             #pylint: disable=no-self-use
67             @classmethod
68             def close(cls) -> None:
69                 """Ferme la connexion courante
70                 """
71                 SqliteController.connection = getattr(g, '_database',
72                 None)
72                 if SqliteController.connection is not None:
73                     SqliteController.connection.close()
74
75             @classmethod
76             def execute_many(cls, query: str, values: tuple) -> None:
77                 """Fait une requete de type many. (INSERT INTO <table
78                 > VALUES(...values), (...values), (...values), ... )
79                 """
80                 cursor = cls.__get_cursor()
81
82                 cursor.executemany(query, values)
83
84                 cls.get_instance().commit()
85
86             @classmethod
87             def execute(cls, query: str, values = None, fetch_mode = 2
88             ) -> Any:
89                 """Exécute une requete en base
90
91
92
93
94

```

```

85             Arguments:
86                 query {str} -- Requete à executer
87
88             Keyword Arguments:
89                 values {tuple} -- Valeurs à utilisé (default
90 : {None})
91                 fetch_mode {int} -- Type de fetch voulu (
92 default: {2}) -> pas de fetch
93
94             Returns:
95                 None/[]/int -- Si pas de fetch, retourne le
96 last_row_id, sinon retourne le fetch voulu
97 """
98
99     try:
100         cursor = cls.__get_cursor()
101
102         if values is not None:
103             cursor.execute(query, values)
104         else:
105             cursor.execute(query)
106
107         last_row_id = cursor.lastrowid
108
109         cls.get_instance().commit()
110
111         if fetch_mode == cls.FETCH_ONE:
112             result = cursor.fetchone()
113         elif fetch_mode == cls.FETCH_ALL:
114             result = cursor.fetchall()
115         else:
116             result = None
117
118         if (fetch_mode == cls.NO_FETCH):
119             return last_row_id
120
121             return result
122     except sqlite3.Error as e:
123         raise e
124
125     # Données de la base de données
126
127     @classmethod
128     def truncate_anime_related(cls) -> None:
129         """Vide toutes les tables relatives aux animes
130         """
131
132         try:
133             cls.execute("DELETE FROM `user_has_favorite`")

```

```

129         cls.execute("DELETE FROM `list_has_anime`")
130         cls.execute("DELETE FROM `anime_ft`")
131         cls.execute("DELETE FROM `anime`")
132         cls.execute("DELETE FROM `url`")
133         cls.execute("DELETE FROM `type`")
134         cls.execute("DELETE FROM `status`")
135
136     return True
137 except SqliteError as e:
138     log(e)
139     return False
140
141 @classmethod
142 def setup_anime_table(cls) -> None:
143     """Créer la table anime si elle n'existe pas
144     """
145     try:
146         cls.execute(
147             """
148                 CREATE TABLE IF NOT EXISTS `anime` (
149                     `idAnime` integer NOT NULL PRIMARY
150                     KEY,
151                     `title` text NOT NULL,
152                     `type` integer NOT NULL,
153                     `episodes` integer NOT NULL,
154                     `status` integer NOT NULL,
155                     `picture` text NOT NULL,
156                     `thumbnail` text NOT NULL,
157                     `synonyms` text DEFAULT NULL,
158                     `modificationDate` timestamp DEFAULT
159                     NULL,
160                     `(``idType``),
161                     FOREIGN KEY(`type`) REFERENCES `type`
162                     `status`(`idStatus`)
163                     )
164                     # Obligatoire pour pouvoir faire une recherche
165                     full text
166                     cls.execute("CREATE VIRTUAL TABLE IF NOT EXISTS `"
167                     `anime_ft` USING FTS5(`idAnime`, `title`)")
168                     return True
169 except SqliteError as e:
170     log(e)
171     return False

```

```

170
171     @classmethod
172     def setup_type_table(cls) -> None:
173         """Créer la table type si elle n'existe pas
174         """
175         try:
176             cls.execute(
177                 """
178                 CREATE TABLE IF NOT EXISTS `type` (
179                     `idType` integer NOT NULL PRIMARY KEY
180                     ,
181                     `nameType` text NOT NULL,
182                     `modificationDate` timestamp DEFAULT
183                     NULL
184                     )
185                     """
186                     ,
187                     None
188                     )
189                     return True
190             except SqliteError as e:
191                 log(e)
192                 return False
193
194             @classmethod
195             def setup_status_table(cls) -> None:
196                 """Créer la status anime si elle n'existe pas
197                 """
198                 try:
199                     cls.execute(
200                         """
201                             CREATE TABLE IF NOT EXISTS `status` (
202                                 `idStatus` integer NOT NULL PRIMARY
203                                 KEY,
204                                 `nameStatus` text NOT NULL,
205                                 `modificationDate` timestamp DEFAULT
206                                 NULL
207                                 )
208                                 """
209                                 ,
210                                 None
211                                 )
212                                 return True
213             except SqliteError as e:
214                 log(e)
215                 return False
216
217             @classmethod
218             def setup_url_table(cls) -> None:

```

```

213         """Créer la url anime si elle n'existe pas
214         """
215     try:
216         cls.execute(
217             """
218             CREATE TABLE IF NOT EXISTS `url` (
219                 `idUrl` integer NOT NULL PRIMARY KEY,
220                 `urlName` text NOT NULL,
221                 `idAnime` integer NOT NULL,
222                 `isRelation` integer NOT NULL,
223                 `modificationDate` timestamp DEFAULT
224                 NULL,
225                 FOREIGN KEY (`idAnime`) REFERENCES `anime`(`idAnime`)
226             )
227             """
228         )
229         return True
230     except SqliteError as e:
231         log(e)
232         return False

233     @classmethod
234     def setup_user_table(cls) -> None:
235         """Créer la user anime si elle n'existe pas
236         """
237     try:
238         cls.execute(
239             """
240             CREATE TABLE IF NOT EXISTS `user` (
241                 `idUser` integer NOT NULL PRIMARY KEY
242                 ,
243                 `emailUser` text NOT NULL,
244                 `password` text NOT NULL,
245                 `nicknameUser` text NOT NULL,
246                 `modificationDate` timestamp DEFAULT
247                 NULL
248             )
249             """
250         )
251         return True
252     except SqliteError as e:
253         log(e)
254         return False

255     @classmethod

```

```

256     def setup_list_table(cls) -> None:
257         """Créer la list anime si elle n'existe pas
258         """
259         try:
260             cls.execute(
261                 """
262                     CREATE TABLE IF NOT EXISTS `list` (
263                         `idList` integer NOT NULL PRIMARY KEY
264                         ,
265                         `nameList` text NOT NULL,
266                         `modificationDate` timestamp DEFAULT
267                         NULL
268                         )
269                         ,
270                         None
271                     )
272                     return True
273                 except SqliteError as e:
274                     log(e)
275                     return False
276
277 @classmethod
278     def setup_user_has_list_table(cls) -> None:
279         """Créer la user_has_list anime si elle n'existe pas
280         """
281         try:
282             cls.execute(
283                 """
284                     CREATE TABLE IF NOT EXISTS `user_has_list`
285                         (
286                             `idUserHasList` integer NOT NULL
287                             PRIMARY KEY,
288                             `user`(`idUser`),
289                             `list`(`idList`),
290                             `modificationDate` timestamp DEFAULT
291                             NULL
292                             )
293                             ,
294                             None
295                         )
296                         )
297                         return True
298                     except SqliteError as e:
299                         log(e)
300                         return False

```

```
296     @classmethod
297     def setup_list_has_anime_table(cls) -> None:
298         """Créer la table list_has_anime si elle n'existe pas
299         """
300         try:
301             cls.execute(
302                 """
303                     CREATE TABLE IF NOT EXISTS `list_has_anime` (
304                         `idListHasAnime` integer NOT NULL
305                         PRIMARY KEY,
306                         `list`(`idList`),
307                         `anime`(`idAnime`),
308                         `modificationDate` timestamp DEFAULT
309                         NULL
310                         )
311                         """
312                         ,
313                         None
314                         )
315                         return True
316                         except SqliteError as e:
317                             log(e)
318                             return False
319
320     @classmethod
321     def setup_user_has_favorite_table(cls) -> None:
322         """Créer la table user_has_favorite si elle n'existe
323         pas
324         """
325         try:
326             cls.execute(
327                 """
328                     CREATE TABLE IF NOT EXISTS `user_has_favorite` (
329                         `idUserHasFavorite` integer NOT NULL
330                         PRIMARY KEY,
331                         `user`(`idUser`),
332                         `anime`(`idAnime`),
333                         `orderId` integer NOT NULL,
334                         `modificationDate` timestamp DEFAULT
335                         NULL
336                         )
337                         """
338                         ,
339                         None
340                         )
341                         return True
342                         except SqliteError as e:
343                             log(e)
344                             return False
```

```

332             None
333         )
334         return True
335     except SqliteError as e:
336         log(e)
337         return False
338
339     # pylint: disable=too-many-locales,too-many-nested-blocks
340     @classmethod
341     def synchronise_with_mysql(cls) -> bool:
342         """Synchronise la base MySQL avec Sqlite3
343
344         Returns:
345             bool -- Status de la synchronisation
346         """
347
348     try:
349         tables = {
350             'type': 'idType',
351             'statusidStatus',
352             'anime': 'idAnime',
353             'url': 'idUrl',
354             'user': 'idUser',
355             'list': 'idList',
356             'user_has_list': 'idUserHasList',
357             'list_has_anime': 'idListHasAnime',
358             'user_has_favorite': 'idUserHasFavorite' ,
359         }
360         sql_table_columns = "PRAGMA table_info(""
361         sql_entries = "SELECT * FROM "
362         sql_insert = "INSERT INTO "
363         sql_delete = "DELETE FROM "
364
365         connection = MySQLController().openConnection()
366         MySQLController().executeQuery(connection, "SET
FOREIGN_KEY_CHECKS=0;", fetch_mode=MySQLController.NO_FETCH,
367         close_connection=False)
368
369         for current_table in tables:
370             print('TABLE COURANTE :', current_table)
371
372             # Récupération des noms de colonnes de la
373             # table courante
374             current_tables_columns = SqliteController().
375             execute(sql_table_columns + current_table + ')', fetch_mode=
376             SqliteController.FETCH_ALL)
377             current_tables_columns = [item['name'] for
378             item in current_tables_columns]

```

```

373
374          # Récupération des enregistrements de la
375          # table courante sur sqlite et mysql
376          # current_table, fetch_mode=SqliteController.FETCH_ALL)
377          # mysql_entries = MySQLController().executeQuery(connection,
378          # sql_entries + current_table,
379          # fetch_mode=MySQLController.FETCH_ALL, close_connection=False)
380
381          # Création d'un tableau contenant l'id et la
382          # modificationDate de chaque enregistrement
383          # sqlite_ids_timestamps = [{id: sqlite_entry[tables[current_table]],
384          # 'timestamp': sqlite_entry['modificationDate'].replace(microsecond=0)} for sqlite_entry
385          # in sqlite_entries]
386          # mysql_ids_timestamps = [({id: mysql_entry[tables[current_table]],
387          # 'timestamp': mysql_entry['modificationDate'].replace(microsecond=0)}) for mysql_entry
388          # in mysql_entries]
389
390          # Tri des tableaux par id
391          # sqlite_ids_timestamps = sorted(
392          # sqlite_ids_timestamps, key=lambda k: k['id'])
393          # mysql_ids_timestamps = sorted(
394          # mysql_ids_timestamps, key=lambda k: k['id'])
395
396          print('Data same', len(sqlite_entries) == len(mysql_entries))
397          ids_to_skip_for_update = []
398          # Ajout des données manquantes mysql et
399          # suppression des données en trop mysql
400          if len(sqlite_entries) != len(mysql_entries):
401              # Ajout des éléments manquants dans mysql
402              ids_non_present_in_mysql = []
403              if len(sqlite_ids_timestamps) > len(mysql_ids_timestamps):
404                  # Éléments non présents dans la table
405                  mysql
406                  ids_non_present_in_mysql = [i['id']
407                  for i in sqlite_ids_timestamps if i['id'] not in [j['id'] for
408                  j in mysql_ids_timestamps]]
409
410          # Récupère les valeurs sqlite à
411          # ajouté dans mysql seulement si elles ne sont déjà présentes
412          # dans mysql
413          values_to_append = []
414          for sqlite_entry in sqlite_entries:

```

```

399                     if sqlite_entry[tables[
400                         current_table]] in ids_non_present_in_mysql:
401                             values_to_append.append(tuple
402                               (sqlite_entry.values()))
403                             ids_to_skip_for_update.append
404                               (sqlite_entry[tables[current_table]])
405
406                             # Construction de la requête d'ajout
407                             # de données et insertion des données dans mysql
408                             columns_list = ''
409                             for column in current_tables_columns
410                               : columns_list += column + ','
411                             parameter_list = ''
412                             for _ in range(len(
413                               current_tables_columns)): parameter_list += '%s,'
414                             # Format de la requête :
415                             #           INSERT INTO <table>(...champs
416                             ) VALUES(...valeurs)
417                             MySQLController().executeMany(
418                               connection,
419                               sql_insert + current_table + "(" + columns_list[:-1] + ")"
420                               VALUES(" + parameter_list[:-1] + ")",
421                               values=values_to_append)
422                             # Suppression des éléments en trop de
423                             # mysql
424                             elif len(sqlite_ids_timestamps) < len(
425                               mysql_ids_timestamps):
426                               # Éléments non présents dans la
427                               # tablea sqlite
428                               ids_non_present_in_sqlite = [i['id']
429                               for i in mysql_ids_timestamps if i['id'] not in [j['id'] for
430                                 j in sqlite_ids_timestamps]]
431
432                               # Construction de la requête de
433                               # suppressions de données
434                               delete_list = ''
435                               for delete in
436                                 ids_non_present_in_sqlite: delete_list += str(delete) + ','
437
438                               # Format de la requête :
439                               #           DELETE FROM <table> WHERE <id
440                               #           table> IN (...valeurs)
441                               MySQLController().executeQuery(
442                                 connection,
443

```

```

425 sql_delete + current_table + " WHERE " + tables[current_table]
    ] + " IN (" + delete_list[:-1] + ")",
426     fetch_mode=MySQLController.NO_FETCH,
427     close_connection=False)
428
429             # Réafectation des variables pour avoir les
430             # données à jour
431             mysql_entries = MySQLController().
432             executeQuery(connection, sql_entries + current_table,
433             fetch_mode=MySQLController.FETCH_ALL, close_connection=False)
434             mysql_ids_timestamps = [({'id': mysql_entry[
435             tables[current_table]], 'timestamp': mysql_entry['
436             modificationDate'].replace(microsecond=0)}) for mysql_entry
437             in mysql_entries]
438             mysql_ids_timestamps = sorted(
439             mysql_ids_timestamps, key=lambda k: k['id'])
440
441             # Récupération des id à mettre à jour dans
442             # mysql
443             elements_to_updates = []
444             for sqlite_element in mysql_ids_timestamps:
445                 if sqlite_element['id'] not in
446                     ids_to_skip_for_update:
447                         # Format de la requete :
448                         #         SELECT <id table> AS `id`  

449                         # FROM <table> WHERE modificationDate NOT LIKE '<date mysql>%'  

450                         # AND <id table> = <id mysql>
451                         # Je suis obligé de mettre le `NOT  

452                         # LIKE` pour la date car Sqlite contient les millisecondes et  

453                         # pas mysql
454                         entry = SqliteController().execute("
455             SELECT " + tables[current_table] + " AS `id` FROM " +
456             current_table + " WHERE modificationDate NOT LIKE '" + str(
457             sqlite_element['timestamp']) + "%' AND " + tables[
458             current_table] + " = " + str(sqlite_element['id']),
459             fetch_mode=SqliteController.FETCH_ONE)
460             if entry is not None:
461                 elements_to_updates.append(entry[
462                     'id'])
463
464             print(elements_to_updates)
465             # A voir avec Bonvin car trop long
466             for index in elements_to_updates:
467                 # Récupération des valeurs sqlite à

```

```

449 mettre dans mysql
450                     values_to_append = None
451                     for item in sqlite_entries:
452                         if item[tables[current_table]] ==
453                             index:
454                                 # [1:] pour éviter de prendre l'
455                                 id
456                                 values_to_append = tuple(list(
457                                     item.values())[1:])
458
459                         break
460
461                     # Mise à jours des données mysql avec les
462                     # valeurs sqlite
463                     set_list = ''
464                     # [1:] pour éviter de mettre à jour l'id
465                     for column in current_tables_columns[1
466 :]: set_list += column + "=%s,"
467                     MySQLController().executeQuery(connection
468 ,
469
470                     "UPDATE
471                     " + current_table + " SET " + set_list[:-1] + " WHERE "
472                     tables[current_table] + " = " + str(index),
473                     values=
474                     values_to_append,
475
476                     fetch_mode=MySQLController.NO_FETCH,
477
478                     close_connection=False)
479                     MySQLController().executeQuery(connection, "SET
480                     FOREIGN_KEY_CHECKS=1;", fetch_mode=MySQLController.NO_FETCH,
481                     close_connection=False)
482                     MySQLController().closeConnection(connection)
483                     return True
484             except (SqliteError, MysqlError, ValueError,
485             TypeError) as e:
486                 log(e)
487                 return False
488
489

```

```

1  """Contient la classe de contrôle des status
2  Les status sont : FINISHED, CURRENTLY, UPCOMING, UNKNOWN
3
4  @author: Tanguy Cavagna
5  @copyright: Copyright 2020, TPI
6  @version: 1.0.0
7  @date: 2020-05-26
8  """
9  from sqlite3 import Error as SqliteError
10 from .SqliteController import SqliteController
11 from ..models.Status import Status
12 from .logger import log
13
14 class StatusController:
15     """Contrôleur des status
16     """
17
18     def __init__(self):
19         pass
20
21     @classmethod
22     def get_all(cls) -> [Status]:
23         """Récupère tout les statuts
24
25             Returns:
26                 [Status] -- Listes contenant les statuts
27             """
28
29         try:
30             rows = SqliteController().execute("SELECT `idStatus`, `nameStatus` FROM `status`", fetch_mode=SqliteController().FETCH_ALL)
31             status = [Status(t['idStatus'], t['nameStatus']) for t in rows]
32
33             return status
34         except SqliteError as e:
35             log(e)
36             raise e
37
38     @classmethod
39     def get_all_as_dict(cls) -> dict:
40         """Récupère tout les statuts sous forme de
41             dictionnaire
42
43             Returns:
44                 dict -- Dictionnaire contenant les statuts
45                 avec le format "name": "id"

```

```
43     """
44     status = {}
45     for s in cls.get_all():
46         status.update({s.name: s.id})
47
48     return status
```

```
1 """Contient la classe de contrôle des activités
2
3 @author: Tanguy Cavagna
4 @copyright: Copyright 2020, TPI
5 @version: 1.1.0
6 @date: 2020-06-02
7 """
8 from datetime import timedelta
9 from datetime import datetime as dt
10 from sqlite3 import Error as SqliteError
11
12 from .SqliteController import SqliteController
13
14 class ActivitiesController:
15     """Controlleur d'une activité
16     """
17
18     def __init__(self):
19         pass
20
21     @classmethod
22     def get_last_24h(cls, user_id: int) -> []:
23         """Récupère les activités d'un utilisateur
24
25             Arguments:
26                 user_id {int} -- Id de l'utilisateur
27             Returns:
28
29                 [] -- Liste des activités
30         """
31         activities = cls.__get_list_related(user_id) + cls.
32         __get_favorite_related(user_id)
33         # Trie toutes les activités par date
34         activities.sort(key=lambda item: item['date'], reverse
35         =True)
36         # Change le format de toutes les dates des activités
37         for a in activities:
38             a['date'] = a['date'].strftime('%Y-%m-%d %H:%M:%S')
39
40     return activities
41
42     @classmethod
43     def __get_list_related(cls, user_id: int) -> []:
44         """Récupère les activités relatives aux listes
```

```

45             Arguments:
46                 user_id {int} -- Id de l'utilisateur
47             Returns:
48
49                 [] -- Liste des activités
50             """
51     try:
52         current_datetime = dt.now().replace(microsecond=0)
53
54         sql_list_additions = """SELECT anime.title, anime.
55 picture, list_has_anime.modificationDate, list.nameList
56             FROM list_has_anime
57             JOIN list ON list.idList
58             = list_has_anime.idList
59             JOIN user_has_list ON
60             user_has_list.idList = list.idList
61             JOIN anime ON anime.
62             idAnime = list_has_anime.idAnime
63             WHERE user_has_list.idUser
64             = ? AND list_has_anime.modificationDate > ?"""
65
66         user_list_additions = SqliteController().execute(
67             sql_list_additions, values=(user_id,(current_datetime -
68             timedelta(days=1))), fetch_mode=SqliteController.FETCH_ALL)
69
70         results = []
71         for add in user_list_additions:
72             results.append({
73                 'title': add['title'],
74                 'list': add['nameList'],
75                 'picture': add['picture'],
76                 'date': add['modificationDate']
77             })
78
79         return results
80     except SqliteError as e:
81         print(str(e))
82         return []
83
84     @classmethod
85     def __get_favorite_related(cls, user_id: int) -> []:
86         """Récupère les activités relatives aux favoris
87
88             Arguments:
89                 user_id {int} -- Id de l'utilisateur
90             Returns:
91
92

```

```
85             [] -- Liste des activités
86             """
87         try:
88             current_datetime = dt.now().replace(microsecond=0
89         )
90             sql_favorite_additions = """SELECT anime.title,
91             anime.picture, user_has_favorite.modificationDate
92                     FROM user_has_favorite
93                     JOIN anime ON anime.
94             idAnime = user_has_favorite.idAnime
95                     WHERE user_has_favorite.
96            idUser = ? AND user_has_favorite.modificationDate > ?"""
97             user_favorite_additions = SqliteController().
98             execute(sql_favorite_additions, values=(user_id,
99             current_datetime - timedelta(days=1))),), fetch_mode=
100             SqliteController.FETCH_ALL)
101             results = []
102             for add in user_favorite_additions:
103                 results.append({
104                     'title': add['title'],
105                     'list': 'favoris',
106                     'picture': add['picture'],
107                     'date': add['modificationDate']
108                 })
109             return results
110         except SqliteError as e:
111             print(str(e))
112             return []
```

```
1  {% extends 'layouts/layout.html' %}  
2  
3  {% block title %}  
4      Aide  
5  {% endblock title %}  
6  
7  {% block content %}  
8      <div class="toc"></div>  
9      <div class="container about-content mt-5">  
10         <h1>Aide</h1>  
11         {{ html | safe }}  
12     </div>  
13  {% endblock content %}  
14  
15  {% block script %}  
16      <script src="{{ url_for('static', filename='js/toc-  
generator.js') }}"></script>  
17  {% endblock script %}
```

```
1  {% extends 'layouts/layout.html' %}  
2  
3  {% block title %}  
4      À propos  
5  {% endblock title %}  
6  
7  {% block content %}  
8      <div class="toc"></div>  
9      <div class="container about-content mt-5">  
10         <h1>À propos</h1>  
11         <p>  
12             Ce projet a été réalisé dans le cadre du <em>  
13             Travail Pratique Individuel</em> (TPI) durant la session de  
14             mai - juin 2020.  
15             Il a pour but de valider les compétences acquises  
16             tout au long de la formation *Informaticien CFC* de l'école du  
17             CFPT-Informatique au Petit-Lancy.  
18             Animanga est une application web, écrite en Python  
19             , permettant aux utilisateurs de faire leur propre  
20             bibliothèque d'anime.  
21             Pour ce faire, l'utilisateur a la possibilité de  
22             créer ses propres listes afin de correctement organiser  
23             sa bibliothèque, mettre des animes en tant que  
24             favoris, et rechercher les animes qu'il voudrait ajouter à sa  
25             collection directement depuis cette application.  
26         </p>  
27         {{ html | safe }}  
28     </div>  
29  {% endblock content %}  
30  
31  {% block script %}  
32      <script src="{{ url_for('static', filename='js/toc-  
33      generator.js') }}"></script>  
34  {% endblock script %}
```

```
1  {% extends 'layouts/layout.html' %}  
2  
3  {% block title %}  
4      Accueil  
5  {% endblock title %}  
6  
7  {% block content %}  
8      <div id="override-info"></div>  
9  
10     {% if not current_user.is_authenticated %}  
11         <div class="background">  
12               
16             <section class="col-md-12 col-xl-8">  
17                 {% if search_results|length <= 0 %}  
18                     <div class="no-search">  
19                         <h4>Aucune recherche effectuée</h4>  
20                         <span class="separator"></span>  
21                         <span>  
22                             Faites une recherche pour obtenir  
des informations sur l'anime souhaité  
23                         </span>  
24                         <span>( · ? ? ? )</span>  
25                     </div>  
26  
27                     <div class="activities"></div>  
28                 {% else %}  
29                     <h4>Résultats pour "{{ search_string }}" :  
30                     </h4>  
31                     <div class="search-results">  
32                         {% for anime in search_results %}  
33                             <div class="result-item">  
34                                 <div class="left">  
35                                     <img class="picture" src  
36                                     = "{{ anime.picture }}>  
37                                     <span class="title" data-  
38                                     toggle="modal" data-target="#result-item-{{ anime.id }}>{{  
39                                         anime.title }}</span>  
40                                 </div>  
41                                 <div class="right">  
42                                     <div class="status">{{  
43                                         anime.status }}</div>  
44                                     <div class="type">{{ anime
```

```

40 .type }}</div>
41 >
42                               {% for relation in
43                                   anime.relations %}
44                               
47                               {% endfor %}
48 </div>
49
50                               <!-- Modal -->
51                               <div class="modal fade" id="
52                                   result-item-{{ anime.id }}" tabindex="-1" role="dialog" aria-
53                                   labelledby="exampleModalLabel" aria-hidden="true">
54                               <div class="modal-dialog"
55                                   modal-dialog-centered" role="document">
56                               <div class="modal-
57                                   content">
58                               <div class="header
59                                   header-background">
60                               <img src
61                                   = "{{ url_for('static', filename='img/home-background.png') }}"
62                               >
63                               <div class
64                                   ="dark-layer"></div>
65
66                               <div class="thumbnail" src="{{ anime.picture }}>
67                               <div class="title">{{ anime.title }}</div>
68                               <div class="favorite-toggler {{ 'is-favorite' if anime.is_favorite else
69                                   '' }}" data-id="{{ anime.id }}>
70                               <svg width
71                                   = "21" height="19" fill="none" xmlns="http://www.w3.org/2000/
72                                   svg">
73                               <path
74                                   d="M19.291 1.612a5.5 5.5 0 00-7.78 0l-1.06 1.06-1.06-1.06a5.
75                                   501 5.501 0 10-7.78 7.78l1.06 1.06 7.78 7.78 7.78-7.78 1.06-1.
76                                   06a5.5 5.5 0 000-7.78z" fill="#fff"/>
77                               </svg>

```

```
64                                     </div>
65                                     </div>
66
67                                     <div class="content">
68                                         <div class="status">
69                                             Status</span>
70                                         <form>
71                                             <div name="status" class="status_combo">
72                                                 <option value="{{ anime.id }}-none">---</option>
73                                                 <%
74                                                 for list in user_list[:4] %>
75                                                 <option value="{{ anime.id }}-{{ list.id }}" {{ 'selected' if
76                                                 list.id in anime.in_list else '' }}>{{ list.name }}</option>
77                                                 <%
78                                                 endfor %>
79                                         </div>
80                                         </form>
81                                         <div class="custom">
82                                             <span>Listes personnelles</span>
83                                         <%
84                                         for list in user_list[4:] %>
85                                         <div class="list-item">
86                                             <div class="custom-check {{ 'checked' if list.id in anime.
87                                                 in_list else '' }}>
88                                                 <input type="checkbox" value="{{ anime.id }}-{{ list.
89                                                 id }}">
90                                         </div>
91                                         <span class="list-title">{{ list.name }}</span>
92                                     </div>
93                                     <%
```

```
89 endfor %}
90
91
92
93
94
95
96
97     {% endfor %}
98
99     {% endif %}
100 </section>
101
102 <aside class="col-md-12 col-xl-4">
103     <h4>Favoris</h4>
104
105     <div class="favorites-container">
106         <div class="favorites"></div>
107     </div>
108 </aside>
109 </main>
110
111 {% endif %}
112
113 {% block script %}
114     <script src="{{ url_for('static', filename='js/landing-
page-handler.js') }}"></script>
115     <script src="{{ url_for('static', filename='js/fetch-
favorites.js') }}"></script>
116     <script src="{{ url_for('static', filename='js/fetch-
activities.js') }}"></script>
117     <script src="{{ url_for('static', filename='js/ratio-
keeper.js') }}"></script>
118     <script src="{{ url_for('static', filename='js/anime-
handler.js') }}"></script>
119     <script>
120         window.addEventListener('load', () => {
121             fetchFavorites(false, false, null); fetchActivities(); });
122
123             // Déclaration obligatoire pour que les message du
124             // stream puissent être mis dans un élément DOM
125             let override_info = document.getElementById('override-
info');
126             </script>
127             <!-- Affiche les messages envoyé par la fonction
128             stream_template -->
129             {% for message in override_messages: %}
```

```
127      <script>
128          // Prise en compte du message spécial 'redirect'
129          // pour rediriger l'utilisateur et de ce fait, arrêter le stream
130          // courant
131          if ("{{ message }}" == "redirect") {
132              override_info.style.padding = '0';
133              window.location.replace("/");
134          }
135          // Condition obligatoire à cause du script ci-
136          // dessus ayant déjà cette variable déclaré
137          if (typeof override_info !== 'undefined') {
138              override_info = document.querySelector('#
139              override-info');
140          } else {
141              let override_info = document.querySelector('#
142              override-info');
143          }
144          // Utilisation de 'var' pour ne pas avoir de
145          // soucis avec des variables déjà déclarées lors de messages
146          // précédents
147          var msg = document.createElement('span');
148          var loader = document.createElement('img');
149
150          override_info.innerHTML = '';
151          override_info.style.padding = '.75rem 0';
152          msg.innerHTML = '{{ message }}';
153          loader.src = '/static/img/loader.svg';
154          loader.classList.add('ml-5');
155          override_info.appendChild(msg);
156          override_info.appendChild(loader);
157      </script>
158  {% endfor %}
159  {% endblock script %}
```

```

1  {% extends 'layouts/layout.html' %} 
2
3  {% block metas %} 
4  <meta id="searched_user" data-nickname="{{ searched_user.
5   nickname }}>
6  {% endblock metas %} 
7
8  {% block title %} 
9   Listes - {{ searched_user.nickname }} 
10 {% endblock title %} 
11
12  {% block content %} 
13   {% include 'layouts/profile-header.html' %} 
14
15  <div class="row mt-4">
16    <div class="col-xl-2 col-md-12 mb-5" id="lists">
17      <div class="lists-container">
18        <div id="search-list">
19          
21          <input type="text" class="search-string"
22            placeholder="Rechercher">
23        </div>
24
25        <h4>Listes</h4>
26
27        <ul>
28          <li class="selected"><span class="
29            list__name">Tous</span></li>
30          {% for list in user_lists %} 
31            {% if loop.index > 4 and current_user.
32              nickname == searched_user.nickname %}
33              <li data-list="{{ list.id }}>
34                <span class="list__name">{{
35                  list.name }}</span>
36                <input type="text" class="d-
37                  none" style="width: 80%" name="new_list_name" id="{{ list.id
38                  }}_new_list_name">
39                <span class="cancel__rename d-
40                  none">?</span>
41                <span class="remove__list">?</span>
42            </li>
43            {% else %}
44              <li data-list="{{ list.id }}><
45                span class="list__name">{{ list.name }}</span></li>
46            {% endif %}

```

```
37          {% endfor %}  
38          {% if current_user.nickname ==  
39              searched_user.nickname %}  
39              <li class="new-list">  
40                  <input type="text" name="new-  
40 list__name" id="new-list__name" placeholder="Nouvelle liste">  
41              </li>  
42          {% endif %}  
43      </ul>  
44  </div>  
45  </div>  
46  <div class="col-xl-10 col-md-12 p-md-5 p-xl-0" id="  
46 animes-container"></div>  
47  </div>  
48 {% endblock content %}  
49  
50 {% block script %}  
51 <script src="{{ url_for('static', filename='js/user-lists-  
51 handler.js') }}></script>  
52 {% endblock script %}
```

```
1  {% extends 'layouts/layout.html' %}  
2  
3  {% block title %}  
4      Connexion  
5  {% endblock title %}  
6  
7  {% block content %}  
8      <div class="container-center">  
9          {% for message in get_flashed_messages() %}  
10             <div class="alert alert-warning" id="flash">  
11                 <button type="button" class="close" data-  
12                   dismiss="alert"></button>  
14                 {{ message }}  
15             </div>  
16         {% endfor %}  
17  
18         <div class="login-form">  
19             <h3>Connexion</h3>  
20  
21             <form action="/login" method="post">  
22                 <fieldset>  
23                     <input type="email" name="email" id="email"  
24                     " placeholder="Email">  
25                     {% if auth_errors.email %}  
26                         <ul id="email-error">  
27                             {% for error in auth_errors.email %}  
28                             <li>{{ error }}</li>  
29                         {% endfor %}  
30                         </ul>  
31                     {% endif %}  
32                 </fieldset>  
33  
34                 <fieldset>  
35                     <input type="password" name="password" id=  
36                     ="password" placeholder="Mot de passe">  
37                     {% if auth_errors.password %}  
38                         <ul id="password-error">  
39                             {% for error in auth_errors.  
40                             password %}<li>{{ error }}</li>  
41                         {% endfor %}  
42                         </ul>  
43                     {% endif %}  
44                 </fieldset>  
45  
46                 <div class="submit">  
47                     <button type="submit">Connexion</button>  
48                     <span>Pas encore inscrit ? <a href="  
49                     signup">Créer un compte</a></span>
```

```
41          </div>
42      </form>
43  </div>
44 </div>
45 {% endblock content %}
```

```
1  {% extends 'layouts/layout.html' %}  
2  
3  {% block content %}  
4      <div class="container-center">  
5          {% for message in get_flashed_messages() %}  
6              <div class="alert alert-warning" id="flash">  
7                  <button type="button" class="close" data-  
8                      dismiss="alert">  
15         <h3>Inscris toi <br> sur Animanga</h3>  
16  
17         <form action="/signup" method="post">  
18             <fieldset>  
19                 <input type="email" name="email" id="email  
" placeholder="Email">  
20                     {% if auth_errors.email %}  
21                         <ul id="email-errors">  
22                             {% for error in auth_errors.email  
%}<li>{{ error }}</li>{% endfor %}  
23                         </ul>  
24                     {% endif %}  
25             </fieldset>  
26                 <input type="text" name="nickname" id="  
nickname" placeholder="Pseudo">  
27                     {% if auth_errors.nickname %}  
28                         <ul id="nickname-errors">  
29                             {% for error in auth_errors.  
nickname %}<li>{{ error }}</li>{% endfor %}  
30                         </ul>  
31                     {% endif %}  
32             </fieldset>  
33  
34             <fieldset>  
35                 <input type="password" name="password" id  
="password" placeholder="Mot de passe">  
36                     {% if auth_errors.password %}  
37                         <ul id="password-errors">  
38                             {% for error in auth_errors.  
password %}<li>{{ error }}</li>{% endfor %}  
39                         </ul>
```

```
40          {%- endif %}  
41      </fieldset>  
42  
43      <fieldset>  
44          <input type="password" name="confirm" id="  
  confirm" placeholder="Confirmer le mot de passe">  
45          {%- if auth_errors.confirm %}  
46              <ul id="confirm-errors">  
47                  {%- for error in auth_errors.  
  confirm %}<li>{{ error }}</li>{%- endfor %}  
48          </ul>  
49          {%- endif %}  
50      </fieldset>  
51  
52      <div class="submit">  
53          <button type="submit">Inscription</button>  
54          <span>Déjà un compte ? <a href="{{ url_for  
  ('auth_bp.login') }}>Connecte toi</a></span>  
55      </div>  
56  </form>  
57</div>  
58</div>  
59 {%- endblock content %}
```

```
1  {% extends 'layouts/layout.html' %}  
2  
3  {% block metas %}  
4  <meta id="searched_user" data-nickname="{{ searched_user.  
nickname }}">  
5  {% endblock metas %}  
6  
7  {% block title %}  
8      {{ current_user.nickname }} - Profil  
9  {% endblock title %}  
10  
11 {% block content %}  
12     {% include 'layouts/profile-header.html' %}  
13  
14     <div class="profile">  
15         <div class="stats">  
16             <div class="watched">  
17                 <span class="count">{{ stats['TV'] }}</span>  
18                 <span class="category">TV Regardé</span>  
19             </div>  
20             <div class="watched">  
21                 <span class="count">{{ stats['Movie'] }}</span>  
22             >  
23                 <span class="category">Films Regardé</span>  
24             </div>  
25             <div class="watched">  
26                 <span class="count">{{ stats['ONA'] }}</span>  
27                 <span class="category">Ona Regardé</span>  
28             </div>  
29             <div class="watched">  
30                 <span class="count">{{ stats['Special'] }}</span>  
31             >  
32                 <span class="category">Speciaux Regardé</span>  
33             </div>  
34         <div class="favorites-container">  
35             <div class="favorites"></div>  
36         </div>  
37     </div>  
38 {% endblock content %}  
39  
40 {% block script %}  
41     <script src="{{ url_for('static', filename='js/fetch-  
favorites.js') }}></script>  
42     <script>  
43         window.addEventListener('load', () => { fetchFavorites
```

```
43 (false, false, document.getElementById('searched_user').  
    dataset.nickname} );  
44     </script>  
45 {% endblock script %}
```

```
1  {% extends 'layouts/layout.html' %}  
2  
3  {% block title %}Titre de la page{% endblock title %}  
4  
5  {% block extra %}  
6  <link rel="stylesheet" type="text/css" href="{{ url_for('  
7  static', filename='css/swagger-ui.css') }}" />  
8  <link rel="icon" type="image/png" href="{{ url_for('static',  
9  filename='img/favicon-32x32.png') }}" sizes="32x32" />  
10 <link rel="icon" type="image/png" href="{{ url_for('static',  
11 filename='img/favicon-16x16.png') }}" sizes="16x16" />  
12 <style>  
13     html  
14     {  
15         box-sizing: border-box;  
16         overflow: -moz-scrollbars-vertical;  
17         overflow-y: scroll;  
18     }  
19  
20     *,  
21     *:before,  
22     *:after  
23     {  
24         box-sizing: inherit;  
25         color: black;  
26     }  
27  
28     body  
29     {  
30         margin:0;  
31         background: #fafafa;  
32     }  
33 </style>  
34 {% endblock extra %}  
35  
36  {% block content %}  
37  <div id="swagger-ui"></div>  
38  {% endblock content %}  
39  
40  {% block script %}  
41  <script src="{{ url_for('static', filename='js/swagger-ui-  
42  bundle.js') }}"> </script>  
43  <script src="{{ url_for('static', filename='js/swagger-ui-  
44  standalone-preset.js') }}"> </script>  
45  <script>  
46  window.onload = function() {  
47      // Begin Swagger UI call region
```

```
43     const ui = SwaggerUIBundle({
44         url: "/specs",
45         dom_id: '#swagger-ui',
46         deepLinking: true,
47         supportedSubmitMethods: [], // Used to show the "try
48         it out" button. Possible item: ["get", "post", "delete", "put"
49         ]
50         presets: [
51             SwaggerUIBundle.presets.apis,
52             SwaggerUIStandalonePreset
53         ],
54         plugins: [
55             SwaggerUIBundle.plugins.DownloadUrl
56         ],
57         layout: "StandaloneLayout"
58     });
59     // End Swagger UI call region
60 }
61 </script>
62 {% endblock script %}
```

```
1  {% extends 'layouts/layout.html' %}  
2  
3  {% block title %}  
4      Listes - {{ searched_user.nickname }}  
5  {% endblock title %}  
6  
7  {% block content %}  
8      {% include 'layouts/profile-header.html' %}  
9  
10     <div class="profile">  
11         <button class="reorder">Réordonner les favoris</button>  
12     >  
13         <div class="favorites-container">  
14             <div class="favorites"></div>  
15         </div>  
16     </div>  
17  {% endblock content %}  
18  
19  {% block script %}  
20      <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.min.js" integrity="sha256-VazP97ZCwtekAsvgPBSUwPFKdrwD3unUfSGVYrahUqU=" crossorigin="anonymous"></script>  
21      <script src="{{ url_for('static', filename='js/fetch-favorites.js') }}"></script>  
22      <script src="{{ url_for('static', filename='js/favorites-reorder.js') }}"></script>  
23      <script>  
24          window.addEventListener('load', () => { fetchFavorites(  
25              false, false); });  
26  </script>  
27  {% endblock script %}
```

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial
6         -scale=1.0">
7     <meta http-equiv="X-UA-Compatible" content="ie=edge">
8
9     {% block metas %}{% endblock metas %}
10
11    <title>{% block title %}{% endblock title %}</title>
12
13    <link rel="stylesheet" href="{{ url_for('static', filename
14        ='css/bootstrap.css') }}>
15    <link rel="stylesheet" href="{{ url_for('static', filename
16        ='css/style.min.css') }}>
17        {% block extra %}{% endblock extra %}
18 </head>
19 <body style="overflow-x: hidden">
20
21     <nav class="navbar navbar-expand-lg">
22         <div class="container">
23             <a class="navbar-brand" href="/">
24                 
26
27                 <button class="navbar-toggler" type="button" data-
28                     toggle="collapse" data-target="#navbarSupportedContent" aria-
29                     controls="navbarSupportedContent" aria-expanded="false" aria-
30                     label="Toggle navigation">
31                     <span class="navbar-toggler-icon"><img src
32                         ="{{ url_for('static', filename='img/burger.svg') }}"></span>
33
34             <div class="collapse navbar-collapse" id="
35                 navbarSupportedContent">
36                 <ul class="navbar-nav mr-auto">
37                     <li class="nav-item active">
38                         <a class="nav-link" href="/">Accueil</
39                         a>
40                         </li>
41
42                         {% if current_user.is_authenticated %}>
43                         <li class="nav-item">
44                             <a class="nav-link" href="/profile
45 /{{ current_user.nickname }}">Profil</a>
46                         </li>
```

```
37          <li class="nav-item">
38              <a class="nav-link" href="/lists
39                  /{{ current_user.nickname }}>Listes</a>
40              </li>
41          {% endif %}
42
43          <li class="nav-item active">
44              <a class="nav-link" href="/about">À
45
46      propos</a>
47          </li>
48      <li class="nav-item active">
49          <a class="nav-link" href="/help">Aide
50      </a>
51          </li>
52      </ul>
53
54      <ul class="navbar-nav">
55          {% if current_user.is_authenticated %}
56              <li class="nav-item" style="display:
57                  flex">
58                  <button type="button" id="
59                      synchronize" class="btn mr-5">
60                      
62                  </button>
63              </li>
64              <li class="nav-item" style="display:
65                  flex">
66                  <button type="button" id="pick-
67                      random" class="btn mr-5">
68                      
70                  </button>
71              </li>
72              <li class="nav-item" style="display:
73                  flex">
74                  <!-- Button trigger modal -->
75                  <button type="button" id="search-
76                      modal-trigger" class="btn mr-5" data-toggle="modal" data-
77                      target="#exampleModal">
78                      
80                  </button>
81
82                  <!-- Modal -->
83                  <div class="modal fade search-
84                      modal" id="exampleModal" tabindex="-1" role="dialog" aria-
85
```

```
69  labelledby="exampleModalLabel" aria-hidden="true">
70          <div class="modal-dialog"
71              role="document">
72                  <div class="modal-content">
73                      <div></div>
74                      <input class="search-string" type="text" name="search-string" placeholder="Rechercher sur Animanga">
75                      <div class="search-clear">
76                          <svg width="21" height="21" fill="none" xmlns="http://www.w3.org/2000/svg"><path d="M17.393 1.834h-14a2 2 0 00-2 2v14a2 2 0 002 2h14a2 2 0 002-2v-14a2 2 0 00-2-2zM7.393 7.834l6 6M13.393 7.834l-6 6" stroke="#fff" stroke-width="2" stroke-linecap="round" stroke-linejoin="round"/></svg>
77                      </div>
78                  </div>
79          </div>
80      </li>
81      <li class="nav-item dropdown">
82          <a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button" data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
83              
84          </a>
85          <div class="dropdown-menu" aria-labelledby="navbarDropdown">
86              <a class="dropdown-item" href="/logout">Déconnexion</a>
87          </div>
88          <a class="dropdown-item" href="/override" style="padding: 0 1rem">
89              Écraser tous les animes</span>
90          </a>
91          </div>
92      </li>
93      {% else %}
94          <li class="nav-item">
95              <a class="nav-link" href="/login">Connexion</a>
```

```
96          </li>
97          <li class="nav-item">
98              <a class="nav-link" href="/signup">
99                  "Inscription</a>
100             {% endif %}
101         </ul>
102     </div>
103 </div>
104 </nav>
105
106 <div>
107     {% block content %}{% endblock content %}
108 </div>
109
110 <script src="https://code.jquery.com/jquery-3.4.1.min.js"
111     integrity="sha256-
CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFlBw8HfCJo=" crossorigin="
anonymous"></script>
112     <script src="https://cdnjs.cloudflare.com/ajax/libs/
popper.js/1.14.7/umd/popper.min.js" integrity="sha384-
U02eT0CpHqdSJQ6hJty5KVphtPhzWj9W01clHTMGa3JDZwrnQq4sF86dIHNDz
0W1" crossorigin="anonymous"></script>
113     <script src="https://stackpath.bootstrapcdn.com/bootstrap/
4.3.1/js/bootstrap.min.js" integrity="sha384-
JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFF/nJGzIxFDsf4x0xIM+
B07jRM" crossorigin="anonymous"></script>
114     <script src="{{ url_for('static', filename='js/search-
handler.js') }}></script>
115     <script src="{{ url_for('static', filename='js/shortcut.
js') }}></script>
116     <script>
117         const syncButton = document.querySelector('#
synchronize');
118         if (syncButton != null) {
119             syncButton.addEventListener('click', () => {
120                 window.location.replace('/sync');
121             });
122         }
123     </script>
124     {% block script %}{% endblock script %}
125 </body>
126 </html>
```

```
1 <div class="profile-header">
2     <div class="header">
3         
5         <div class="overlay"></div>
6         
8         <span class="profile-nickname">{{ searched_user.
9             nickname }}</span>
10    </div>
11
12    <div class="links">
13        <span><a href="/profile/{{ searched_user.nickname }}">
14            Overview</a></span>
15        <span><a href="/lists/{{ searched_user.nickname }}">
16            Listes</a></span>
17            {% if current_user.nickname == searched_user.nickname
18            %}
19                <span><a href="/favorites/{{ searched_user.
20                    nickname }}">Favoris</a></span>
21            {% endif %}
22        </div>
23    </div>
```