Dossier de programmation
Création du site web – Société Netflux
A L'INTENTION DE M. BROISIN

Paul CARRÈRE
Paul CERJAK
Youssef OUBENAMMOU
Quentin VUILLIER

Projet Tuteuré
2017-2018
LP GTID

Table des matières

1. Affichage des séries 4
1. Fonction index 4
2. Fonction getSeriesHomeLogged 4
3. Fonction getSeriesHome 5
2. Notation séries 5
1. Affichage des mieux notées 5
2. Fonctions getSeriesTopLogged | getSeriesTop 6
3. Fonction affichage des dernières séries réalisées 6
4. Fonctions getSeriesLastLogged | getSeriesLast 7
5. Fonction affichage des recommandations 7
6. Fonction getInterests 8
7. Fonction getAdvise 8
8. Fonction getUniqueAdvise 9
3. Gestion des favoris 9
1. Récupération des favoris 9
2. Fonction getFavorites 9
4. Fonctions de recherche 10
1. Récupérer les séries suite au champ de recherche 10
2. Fonction getSearchResults 10
3. Fonction getSearchResultsLogged 11
5. Partie administrateur 12
1. Fonction store 12
2. Fonction update 13
3. Fonction destroy 14

4. Récupération des mots-clés

```
// remplir map de mots clés
public static void parcourirExtraireV2(File path) {
    motSerie=new HashMap<String,String>();
    motsScore=new HashMap<String,Integer>();
    dictionnaire=new HashMap<String,Integer>();
    int occurence = 800;
    File[] files = path.listFiles();
    for (File file : files) {
        // Parcours des fichiers
        if (file.getAbsolutePath().endsWith(".srt")|file.getAbsolutePath().endsWith(".sub")) {
            try (BufferedReader br = new BufferedReader(new FileReader(file.getAbsolutePath()))) {
                String sCurrentLine;
                // Parcours des fichiers srt
                while ((sCurrentLine = br.readLine()) != null) {
                    if (!sCurrentLine.isEmpty()) {
                        String str = sCurrentLine.replaceAll("\\pP|0|1|2|3|4|5|6|7|8|9|>|=|<|","," "
                        String line = str.toLowerCase().trim();
                        String[] keywords = line.split(" ");
                        // on enlève les stopwords c'est à dire les mots
                        for (String z: keywords) {
                            if(!stopWordSet.contains(z)){
                                if (!z.isEmpty()){
                                    if(z.length()>2){
                                        if(!motsScore.containsKey(z)){
                                            motsScore.put(z.toLowerCase(),1);
                                        }else{
                                            motsScore.put(z.toLowerCase(),motsScore.get(z)+1);
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        if (file.listFiles() != null) {
            parcourirExtraireV2(file);
        }
```

..... 15

1. Affichage des séries

1. Fonction index

Permet de récupérer la liste des séries en fonction du type d'utilisateur (connecté ou non).

```php
public function index(){

    if(Auth::id()){
        $user_id = (int)Auth::user()->id;
        $series = Serie::getSeriesHomeLogged($user_id);
    }else{
        $series = Serie::getSeriesHome();
    }

    $currentPage = LengthAwarePaginator::resolveCurrentPage();
    $perPage = 12;
    $path=LengthAwarePaginator::resolveCurrentPath();
    $series = new LengthAwarePaginator(array_slice($series, offset: $perPage * ($currentPage - 1), $perPage), count($series), $perPage, $currentPage,['path'=>$path]);

    return view( view: 'home', compact( varname: 'series'));
}
```

Figure 1 : HomeController->Affichage de l'accueil

2. Fonction getSeriesHomeLogged

Récupération des séries et des informations les concernant en fonction de *l'id-utilisateur (Nom série, id_série, lien de l'image, moyenne, note.*

```php
public static function getSeriesHomeLogged($user_id){
    $series = DB::SELECT("(SELECT s.name as name, s.id_Serie as id_Serie, s.image_link as image_link, cast(round(2*avg(notes.note))/2 as decimal(2,1)) as moyenne,
                        ROUND((SELECT distinct no.note FROM notes no WHERE no.id_Notes_User = " . $user_id . " and no.id_Notes_Serie = s.id_Serie), 1) as note

                        FROM series as s, notes as notes
                        WHERE s.id_Serie = notes.id_Notes_Serie
                        GROUP BY s.name, s.image_link, s.id_Serie
                        ORDER BY s.name)
                        union
                        (SELECT se.name as name, se.id_Serie as id_Serie, se.image_link as image_link, 3 as moyenne, 0 as note
                        FROM series se
                        WHERE se.id_Serie not in (SELECT  distinct notes.id_Notes_Serie as id_Serie
                                                  FROM notes )
                        GROUP BY se.name, se.image_link, se.id_Serie)
                        ORDER BY name");
    return $series;
}
```

Figure 2: Serie(model)->getSeriesHomeLogged

3. Fonction getSeriesHome

Fonction de récupération fonctionnant sur le même principe que la précédente mais ne prenant pas en compte la note (car non connecté).

```php
public static function getSeriesHome(){
    $series = DB::SELECT("(SELECT s.name as name, s.id_Serie as id_Serie, s.image_link as image_link, cast(round(2*avg(notes.note))/2 as decimal(2,1)) as moyenne
                        FROM series as s, notes as notes
                        WHERE s.id_Serie = notes.id_Notes_Serie
                        GROUP BY s.name, s.image_link, s.id_Serie
                        ORDER BY s.name)
                        union
                        (SELECT se.name as name, se.id_Serie as id_Serie, se.image_link as image_link, 3 as moyenne
                        FROM series as se
                        WHERE se.id_Serie not in (SELECT  distinct notes.id_Notes_Serie as id_Serie
                                                FROM notes )
                        GROUP BY se.name, se.image_link, se.id_Serie
                        ORDER BY se.name)");
    return $series;
}
```

Figure 3: Serie(model)->getSeriesHome

2. Notation séries

1. Affichage des mieux notées

Récupération des séries ayant une moyenne supérieure à 3.5.

```php
public function index()
{
    if(Auth::id()){
        $user_id = (int)Auth::user()->id;
        $top = Serie::getSeriesTopLogged($user_id);
    }else{
        $top = Serie::getSeriesTop();
    }

    $currentPage = LengthAwarePaginator::resolveCurrentPage();
    $perPage = 12;
    $path=LengthAwarePaginator::resolveCurrentPath();
    $series = new LengthAwarePaginator(array_slice($top,  offset: $perPage * ($currentPage - 1), $perPage), count($top), $perPage, $currentPage,['path'=>$path]);

    return view( view: 'top', compact(  varname: 'series'));
}
```

Figure 4: TopController->Affichage des mieux notées

2. Fonctions getSeriesTopLogged | getSeriesTop

Sur le même principe que précédemment on dispose d'une fonction suivant le type d'utilisateur. Le fait de posséder deux fonctions se produit également pour le listing des dernières séries réalisées et de la recherche.

```php
public static function getSeriesTopLogged($user_id){
    $series = DB::select("(SELECT s.name as name, s.image_link as image_link, s.id_Serie as id_Serie,
                        cast(round(2*avg(notes.note))/2 as decimal(2,1)) as moyenne,
                         ROUND((SELECT distinct no.note FROM notes no WHERE no.id_Notes_User = " . $user_id . " and no.id_Notes_Serie = s.id_Serie), 1) as note
                        FROM notes notes, series s
                        WHERE s.id_Serie = notes.id_Notes_Serie
                        GROUP BY name, id_Serie, image_link
                        HAVING avg(notes.note)>3.5
                        ORDER BY moyenne desc)
                        ");
    return $series;
}

public static function  getSeriesTop(){
    $series = DB::select("(SELECT s.name as name, s.image_link as image_link, s.id_Serie as id_Serie,
                        cast(round(2*avg(notes.note))/2 as decimal(2,1)) as moyenne
                        FROM notes notes, series s
                        WHERE s.id_Serie = notes.id_Notes_Serie
                        GROUP BY s.image_link, s.name, s.id_Serie
                        HAVING avg(notes.note)>3.5
                        ORDER BY moyenne desc)");
    return $series;
}
```

Figure 5: Serie(model)->getSeriesTopLogged & getSeriesTop

3. Fonction affichage des dernières séries réalisées

```php
public function index()
{
    if(Auth::id()) {
        $user_id = (int)Auth::user()->id;
        $last = Serie::getSeriesLastLogged($user_id);
    }else{
        $last = Serie::getSeriesLast();
    }

    $currentPage = LengthAwarePaginator::resolveCurrentPage();
    $perPage = 12;
    $path=LengthAwarePaginator::resolveCurrentPath();
    $series = new LengthAwarePaginator(array_slice($last, offset: $perPage * ($currentPage - 1), $perPage), count($last), $perPage, $currentPage,['path'=>$path]);

    return view( view: 'last', compact( varname: 'series'));
}
```

Figure 6: LastController->Affichage des dernières séries réalisées

4. Fonctions getSeriesLastLogged | getSeriesLast

```php
public static function getSeriesLastLogged($user_id){
    $series = DB::select("SELECT tab.name name, tab.id_Serie id_Serie, tab.image_link image_link, tab.moyenne moyenne, tab.note, tab.release_date from ((
                          SELECT s.name as name, s.id_Serie as id_Serie , s.image_link as image_link,cast(round(2*avg(notes.note))/2 as decimal(2,1)) as moyenne,
                          ROUND((SELECT distinct no.note FROM notes no WHERE no.id_Notes_User = " . $user_id . " and no.id_Notes_Serie = s.id_Serie), 1) as note,
                          s.release_date release_date
                         FROM series s, notes notes
                         WHERE s.id_Serie = notes.id_Notes_Serie
                         GROUP BY s.name, s.id_Serie, s.image_link, s.release_date )

                         union
                          (SELECT se.name as name, se.id_Serie as id_Serie, se.image_link as image_link, 3 as moyenne, 0 as note, se.release_date release_date
                          FROM series as se
                          WHERE se.id_Serie not in (SELECT  distinct notes.id_Notes_Serie as id_Serie
                                                    FROM notes )
                          GROUP BY se.name, se.id_Serie, se.image_link, se.release_date )) as tab
                          ORDER BY tab.release_date desc ");
    return $series;
}

public static function getSeriesLast(){
    $series = DB::select("SELECT tab.name name, tab.id_Serie id_Serie, tab.image_link image_link, tab.moyenne moyenne, tab.release_date release_date from ( (
                         SELECT s.name as name, s.id_Serie as id_Serie , s.image_link as image_link,cast(round(2*avg(notes.note))/2 as decimal(2,1)) as moyenne,
                         s.release_date release_date
                         FROM series s, notes notes
                         WHERE s.id_Serie = notes.id_Notes_Serie
                         GROUP BY s.name, s.image_link, s.id_Serie, s.release_date)
                         union
                          (SELECT se.name as name, se.id_Serie as id_Serie, se.image_link as image_link, 3 as moyenne, se.release_date
                          FROM series se
                          WHERE se.id_Serie not in (SELECT  distinct notes.id_Notes_Serie as id_Serie
                                                    FROM notes )
                          GROUP BY se.name, se.image_link, se.id_Serie, se.release_date )) as tab
                          ORDER BY tab.release_date desc ");
    return $series;
}
```

Figure 7: Serie(model)->getSeriesLastLogged & getSeriesLast

5. Fonction affichage des recommandations

Pour effectuer la recommandation nous commençons par récupérer la liste des séries qui sont aimées et auxquelles l'utilisateur a au minimum donné 4 étoiles.

```php
public function index(){
    $data=[];
    $unique_data=[];
    $user_id = (int)Auth::user()->id;

    $interestedInto = Serie::getInterests($user_id);
    $interest = $this->createList($interestedInto, column: 'name');

    if(count($interestedInto)) {
        $data = Serie::getAdvise($user_id, $interest);
    }

        foreach ($data as $item=>$value){
            array_push( &array: $unique_data, $value->id_Serie);
        }
        $unique_data = array_unique($unique_data);
        $ids = $this->createListWithoutColumn($unique_data);

        $data = Serie::getUniqueAdvise($user_id,$ids);

        $currentPage = LengthAwarePaginator::resolveCurrentPage();
        $perPage = 12;
        $path = LengthAwarePaginator::resolveCurrentPath();
        $series = new LengthAwarePaginator(array_slice($data, offset: $perPage * ($currentPage - 1), $perPage), count($data), $perPage, $currentPage, ['path' => $path]);


    return view( view: 'advise', compact( varname: 'series'));
}
```

Figure 8: AdviseController->Affichage des recommandations

6. Fonction getInterests

```php
public static function getInterests($user_id){
    $series = DB::SELECT("(SELECT distinct s.name FROM series s, favorites f WHERE s.id_Serie = f.id_Favorite_Serie AND f.id_Favorite_User = ".$user_id." )
                UNION(SELECT distinct s.name FROM series s, notes n WHERE s.id_Serie = n.id_Notes_Serie AND n.id_Notes_User = ".$user_id."
                AND n.note >= 4)");
    return $series;
}
```

Figure 9:Serie(model)->getInterests

7. Fonction getAdvise

A partir de cette liste de série on recherche les similarités entre les mots clefs qu'elles contiennent.

```php
public static function getAdvise($user_id, $interests){
    $series = DB::select("with numerator as (
                        SELECT distinct pFAV.id_Post_Serie as idFavoriteSerie, pOther.id_Post_Serie as idOtherSerie,
                        sum(pFAV.term_Frequency * k.idf * pOther.term_Frequency * k.idf) as numValue
                        FROM posting pFAV, posting pOther, keywords k
                        WHERE pFAV.id_Post_Keyword = pOther.id_Post_Keyword
                        AND pFAV.id_Post_Keyword = k.id_Word
                        AND pFAV.id_Post_Serie <> pOther.id_Post_Serie
                        AND pFAV.id_Post_Serie  in (SELECT id_Serie
                                                    FROM series
                                                    WHERE name in " . $interests . ")
                        GROUP BY pFAV.id_Post_Serie, pOther.id_Post_Serie)

                       (SELECT distinct s.id_Serie as id_Serie, s.name as name, n.numValue / (
                                                    sqrt((SELECT sum(power(term_Frequency * idf, 2))
                                                          FROM posting p, keywords k
                                                          WHERE p.id_Post_Keyword = k.id_word
                                                          AND p.id_Post_Serie = n.idFavoriteSerie))
                                                    *
                                                    sqrt((SELECT sum(power(term_Frequency * idf, 2))
                                                          FROM posting p, keywords k
                                                          WHERE p.id_Post_Keyword = k.id_word
                                                          AND p.id_Post_Serie = n.idOtherSerie))) score,
                                                    cast(round(2*avg(notes.note))/2 as decimal(2,1)) as moyenne,
                                                    s.image_link as image_link,
                                                    ROUND((SELECT distinct no.note FROM notes no WHERE no.id_Notes_User = " . $user_id . " and no.id_Notes_Serie = s.id_Serie), 1)
                                                    as note

                        FROM numerator n, series s, notes
                        WHERE n.idOtherSerie = s.id_Serie
                        AND s.id_Serie = notes.id_Notes_Serie
                        AND s.name not in ".$interests."
                        GROUP BY s.name, s.image_link, score, id_Serie
                        ORDER BY score DESC)
                        UNION
                        (SELECT distinct s.id_Serie as id_Serie, s.name as name, n.numValue / (
                                                    sqrt((SELECT sum(power(term_Frequency * idf, 2))
                                                          FROM posting p, keywords k
                                                          WHERE p.id_Post_Keyword = k.id_word
                                                          AND p.id_Post_Serie = n.idFavoriteSerie))
                                                    *
                                                    sqrt((SELECT sum(power(term_Frequency * idf, 2))
                                                          FROM posting p, keywords k
                                                          WHERE p.id_Post_Keyword = k.id_word
                                                          AND p.id_Post_Serie = n.idOtherSerie))) score,
                                                    3 as moyenne,
                                                    s.image_link as image_link,
                                                    0 as note

                        FROM numerator n, series s
                        WHERE n.idOtherSerie = s.id_Serie
                        AND s.id_Serie not in (SELECT  distinct notes.id_Notes_Serie as id_Serie
                                               FROM notes )
                        AND s.name not in ".$interests."
                        GROUP BY s.name, s.image_link, score, id_Serie
                        ORDER BY score DESC)
                        ");

    return $series;
}
```

Figure 10:Serie(model)->getAdvise

8. Fonction getUniqueAdvise

On récupère ensuite les séries en un seul exemplaire.

```
public static function getUniqueAdvise($user_id, $ids){
      $series = DB::SELECT("(SELECT distinct s.name as name, s.id_Serie as id_Serie, s.image_link as image_link, cast(round(2*avg(notes.note))/2 as decimal(2,1)) as moyenne,
                        ROUND((SELECT distinct no.note
                                    FROM notes no
                                    WHERE no.id_Notes_User = ".$user_id."
                                    AND no.id_Notes_Serie = s.id_Serie), 1) as note
                  FROM series as s, notes as notes
                  WHERE s.id_Serie = notes.id_Notes_Serie
                  AND s.id_Serie in ".$ids."
                  GROUP BY s.name, s.image_link, s.id_Serie
                  ORDER BY s.name)
                  union
                  (SELECT distinct se.name as name, se.id_Serie as id_Serie, se.image_link as image_link, 3 as moyenne, 0 as note
                  FROM series as se
                  WHERE se.id_Serie not in (SELECT  distinct notes.id_Notes_Serie as id_Serie
                                    FROM notes )
                                    AND se.id_Serie in ".$ids."
                  GROUP BY se.name, se.image_link, se.id_Serie
                  ORDER BY se.name)");
      return $series;
}
```

Figure 11:Serie(model)->getUniqueAdvise

3. Gestion des favoris

1. Récupération des favoris

Avec les fonctions suivantes nous pouvons récupérer les favoris d'un utilisateur avec pour chaque série la moyenne et la note donnée par l'utilisateur.

```
public function index(){
    $user_id = (int)Auth::user()->id;
    $series = Serie::getFavorites($user_id);

    $currentPage = LengthAwarePaginator::resolveCurrentPage();
    $perPage = 12;
    $path=LengthAwarePaginator::resolveCurrentPath();
    $series = new LengthAwarePaginator(array_slice($series, offset: $perPage * ($currentPage - 1), $perPage), count($series), $perPage, $currentPage,['path'=>$path]);

    return view( view: 'favorites', compact( varname: 'series'));
}
```

Figure 12:FavoritesController->Récupération des favoris

2. Fonction getFavorites

```
public  static function getFavorites($user_id){
    $series = DB::SELECT("(SELECT s.name as name, s.id_Serie as id_Serie, s.image_link as image_link, cast(round(2*avg(notes.note))/2 as decimal(2,1)) as moyenne,
                        ROUND((SELECT distinct no.note
                              FROM notes no
                              WHERE no.id_Notes_User = ".$user_id."
                              AND no.id_Notes_Serie = s.id_Serie), 1) as note
                  FROM series s, favorites f, notes notes
                  WHERE s.id_Serie = f.id_Favorite_Serie
                  AND notes.id_Notes_Serie = f.id_Favorite_Serie
                  AND f.id_Favorite_User = ".$user_id."
                  AND notes.id_Notes_User = f.id_Favorite_User
                  GROUP BY s.name, id_Serie, image_link)
                  UNION
                  (SELECT s.name as name, s.id_Serie as id_Serie, s.image_link as image_link, 3 as moyenne,0 as note
                  FROM series s, favorites f
                  WHERE s.id_Serie = f.id_Favorite_Serie
                  AND f.id_Favorite_User = ".$user_id."
                  AND s.id_Serie not in (SELECT  distinct notes.id_Notes_Serie as id_Serie
                                    FROM notes )
                  GROUP BY name, id_Serie, image_link)
                  ");
    return $series;
}
```

Figure 13:Serie(model)->getFavorites

4. Fonctions de recherche

Pour effectuer une recherche on commence par récupérer la saisie de l'utilisateur que l'on éclate en fonction des espaces et que l'on place dans un tableau. Ensuite pour chaque élément du tableau on le concatène dans une chaîne pour ensuite l'utiliser dans une requête dans une clause « in ».

1. Récupérer les séries suite au champ de recherche

```
public function index(Request $request){
    $query = explode( delimiter: " ", $request->get( key: 's'));

    $in = "";
    for($i =0; $i < count($query) ;$i++){
        $in = $in ."'". $query[$i] ."'". ',';
    }
    $in = substr($in, start: 0, length: -1);

    if(Auth::id()) {
        $user_id = (int)Auth::user()->id;
        $series = Serie::getSearchResultsLogged($user_id, $in);
    }else{
        $series = Serie::getSearchResults($in);
    }


    $currentPage = LengthAwarePaginator::resolveCurrentPage();
    $perPage = 12;
    $path=LengthAwarePaginator::resolveCurrentPath();
    $series= new LengthAwarePaginator(array_slice($series, offset: $perPage * ($currentPage - 1), $perPage), count($series), $perPage, $currentPage,
        ['path'=>$path, 'query'=> $request->query()
        ]);


    return view( view: 'results', compact( varname: 'series'));
}
```

Figure 14:SearchController->Récupérer les séries suite au champ de recherche

2. Fonction getSearchResults

```
public static function getSearchResults($in){
    $series = DB::select("(SELECT s.id_Serie as id_Serie, s.name, s.image_link as url, sum(p.term_Frequency * k.idf) as score,
                          cast(round(2*avg(notes.note))/2 as decimal(2,1)) as moyenne
                          FROM posting p, series s, keywords k, notes notes
                          WHERE p.id_Post_Serie = s.id_Serie
                          AND p.id_Post_Keyword = k.id_Word
                          AND notes.id_Notes_Serie = s.id_Serie
                          AND k.libelle in (". $in .")
                          GROUP BY s.id_Serie , s.name,s.image_link
                          ORDER BY 2 DESC, 1)
                          UNION
                          (SELECT s.id_Serie as id_Serie, s.name, s.image_link as url, sum(p.term_Frequency * k.idf) as score,
                          3 as moyenne
                          FROM posting p, series s, keywords k, notes notes
                          WHERE p.id_Post_Serie = s.id_Serie
                          AND p.id_Post_Keyword = k.id_Word
                          AND k.libelle in (". $in .")
                          AND s.id_Serie not in (SELECT  distinct notes.id_Notes_Serie as id_Serie
                                                 FROM notes )
                          GROUP BY s.id_Serie , s.name,s.image_link
                          ORDER BY 2 DESC, 1)");
    return $series;
}
```

Figure 15:Serie(model)->getSearchResults

3. Fonction getSearchResultsLogged

```
public static function getSearchResultsLogged($user_id, $in){
    $series = DB::select("(SELECT s.id_Serie as id_Serie, s.name, s.image_link as url, sum(p.term_Frequency * k.idf) as score,

                          cast(round(2*avg(notes.note))/2 as decimal(2,1)) as moyenne,
                          ROUND((SELECT distinct no.note FROM notes no WHERE no.id_Notes_User = " . $user_id . " and no.id_Notes_Serie = s.id_Serie), 1) as note
                          FROM posting p, series s, keywords k, notes notes
                          WHERE p.id_Post_Serie = s.id_Serie
                          AND p.id_Post_Keyword = k.id_Word
                          AND notes.id_Notes_Serie = s.id_Serie
                          AND k.libelle in (". $in .")
                          GROUP BY s.id_Serie , s.name,s.image_link
                          ORDER BY 2 DESC, 1)
                          UNION
                          (SELECT s.id_Serie as id_Serie, s.name, s.image_link as url, sum(p.term_Frequency * k.idf) as score,
                          3 as moyenne, 0 as note
                          FROM posting p, series s, keywords k
                          WHERE p.id_Post_Serie = s.id_Serie
                          AND p.id_Post_Keyword = k.id_Word
                          AND k.libelle in (". $in .")
                          AND s.id_Serie not in (SELECT  distinct notes.id_Notes_Serie as id_Serie
                    FROM notes )
                          GROUP BY s.id_Serie , s.name,s.image_link
                          ORDER BY 2 DESC, 1)");
    return $series;
}
```

Figure 16:Serie(model)->getSearchResultsLogged

5. Partie administrateur

Concernant la partie administrateur on retrouve la liste des séries avec quelques informations qui changent de l'interface publique.

Notamment le système de notation, de favoris qui sont remplacés par les boutons « Modifier » et « Supprimer ». Une option d'ajout de série est également présente dans le menu.

Dans la fonction suivante (store) on récupère les champs saisis par l'utilisateur ainsi que les fichiers.

Pour l'image on remplace son nom par le nom de la série où les espaces sont remplacés par des « _ ». La même transformation est réalisée pour l'archive des sous-titres. L'image est ensuite copiée dans 2 dossiers différents :

- Le dossier « images » de la partie admin
- Le dossier « images » de la partie publique

Les sous titres sont quant à eux déplacé dans le dossier « temp ».

1. Fonction store

```php
public function store(Request $request){
    request()->validate([
        'name' => 'required',
        'release_date' => 'required',
        'author' => 'required',
        'synopsis' => 'required'
    ]);


    $serie = new Serie();
    $serie->name = $request->name;
    $serie->author = $request->author;
    $serie->release_date = $request->release_date;
    $serie->synopsis = $request->synopsis;

    //Traitement de l'image
   if($request->hasFile( key: 'image')) {
        $file = $request->file( key: 'image');
        $name = str_replace( search: ' ', replace: '_', strtolower($request->name)).".".$file->getClientOriginalExtension();

        $request->file( key: 'image')->move(public_path( path: "images/"), $name);

         copy( source: public_path( path: "images/").$name, dest: '/srv/http/netflux/public/images/'.$name);
        $serie->image_link = '/images/' . $name;
    }


   //Traitement archive de sous-titres
    if($request->hasFile( key: 'subtitles')) {
        $file = $request->file( key: 'subtitles');
        $name = str_replace( search: ' ', replace: '_', strtolower($request->name)).".".$file->getClientOriginalExtension();

        $request->file( key: 'subtitles')->move(public_path( path: "temp/"), $name);

        exec( command: "java -jar WaveAppender.jar " .public_path( path: "temp/"). $name);
    }


    $serie->save();
    $this->setIDF();

    return redirect()->route( route: 'home')
        ->with('success','Série crée avec succès');
}
```

Figure 17:SerieController->Enregistrement de la nouvelle série

Lorsque l'on choisit l'option modifier, le formulaire préremplit est affiché. Toutes les données sont ensuite changées dans la fonction suivant.

2. Fonction update

```php
public function update(Request $request, $id)
{
    request()->validate([
        'name' => 'required',
        'release_date' => 'required',
        'author' => 'required',
        'synopsis' => 'required'
    ]);
    $dest = public_path( path: "images/");
    $uploaddirpublic = '/srv/http/netflux/public/images/';


    Serie::find($id)->update($request->all());

    if($request->hasFile( key: 'image')) {
        $file = $request->file( key: 'image');
        $name = str_replace( search: ' ', replace: '_', strtolower($request->name)).".".$file->getClientOriginalExtension();

        $request->file( key: 'image')->move($dest, $name);
        copy( source: $dest.$name,  dest: $uploaddirpublic.$name);
        $image_link = '/images/' . $name;


        Serie::find($id)->update(['name'=>$request->name,
            'release_date'=>$request->release_date,
            'author'=>$request->author,
            'synopsis'=>$request->synopsis,
            'image_link'=>$image_link]);

    }else{
        Serie::find($id)->update(['name'=>$request->name,
                                  'release_date'=>$request->release_date,
                                  'author'=>$request->author,
                                  'synopsis'=>$request->synopsis
                                  ]);
    }


    $this->setIDF();

    return redirect()->route( route: 'home')
        ->with('success','Série mise à jour avec succès');
}
```

Figure 18:SerieController->Mise à Jour d'une série

Lors de la suppression d'une série tout ce qui la concerne (commentaires, notes, favoris, mots clefs) sont supprimés.

3. Fonction destroy

```php
public function destroy($id)
{
    if(Comment::where('id_Comment_Serie',$id)->count()>0){
        Comment::where('id_Comment_Serie',$id)->delete() ;
    }
    if( Note::where('id_Notes_Serie',$id)->count()>0){
        Note::where('id_Notes_Serie',$id)->delete();
    }
    if(Favorite::where('id_Favorite_Serie',$id)->count()>0){
        Favorite::where('id_Favorite_Serie',$id)->delete();
    }

    if(Posting::where('id_Post_Serie',$id)->count()>0){
       Posting::where('id_Post_Serie',$id)->delete();
    }

    Serie::find($id)->delete();

    return redirect()->route( route: 'home')
        ->with('success','Série supprimée avec succès');
}
```

Figure 19: SerieController->Suppression d'une série

4. Récupération des mots-clés

```
// remplir map de mots clés
public static void parcourirExtraireV2(File path) {
    motSerie=new HashMap<String,String>();
    motsScore=new HashMap<String,Integer>();
    dictionnaire=new HashMap<String,Integer>();
    int occurence = 800;
    File[] files = path.listFiles();
    for (File file : files) {
        // Parcours des fichiers
        if (file.getAbsolutePath().endsWith(".srt")|file.getAbsolutePath().endsWith(".sub")) {
            try (BufferedReader br = new BufferedReader(new FileReader(file.getAbsolutePath()))) {
                String sCurrentLine;
                // Parcours des fichiers srt
                while ((sCurrentLine = br.readLine()) != null) {
                    if (!sCurrentLine.isEmpty()) {
                        String str = sCurrentLine.replaceAll("\\pP|0|1|2|3|4|5|6|7|8|9|>|=|<|","," "
                        String line = str.toLowerCase().trim();
                        String[] keywords = line.split(" ");
                        // on enlève les stopwords c'est à dire les mots
                        for (String z: keywords) {
                            if(!stopWordSet.contains(z)){
                                if (!z.isEmpty()){
                                    if(z.length()>2){
                                        if(!motsScore.containsKey(z)){
                                            motsScore.put(z.toLowerCase(),1);
                                        }else{
                                            motsScore.put(z.toLowerCase(),motsScore.get(z)+1);
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        if (file.listFiles() != null) {
            parcourirExtraireV2(file);
        }
```

```
    }

    do{
        for (Map.Entry<String, Integer> entry : motsScore.entrySet()) {
            if(entry.getValue()>occurence){
                dictionnaire.put(entry.getKey(), entry.getValue());
                motSerie.put(entry.getKey(), mainptut.zip.replaceFirst(".*/(\\w+).*","$1").toLowerCase());
            }
        }
        occurence-=5;
    }while(dictionnaire.size()<30);

    //      dictionnaireVF.forEach((k, v) -> vf.merge(k, v, Integer::sum));
    //      dictionnaireVO.forEach((k, v) -> vo.merge(k, v, Integer::sum));

    System.out.println(path.getAbsolutePath());
    System.out.println("-------------------------------------");
    System.out.println(dictionnaire);
    System.out.println("La taille du dictionnaire est : "+dictionnaire.size());
}
```