

20/01/2019

TD 4

« J'atteste que ce travail est original, qu'il indique de façon appropriée tous les emprunts, et qu'il fait référence de façon appropriée à chaque source utilisée »

Table des matières

Introduction.....	2
Partie I : Redis Data-store.....	2
Prise en main	2
Question 01	2
Question 02	2
Partie II : Architecture	3
Création d'une infrastructure Docker Node + Redis	3
Question 01	3
Import des données	4
Question 01 et 02	4
Question 03 et 04	5
Portage du code	8
Add Image or Album.....	10
Modify Album.....	12
Modify Image	13
Remove Album	14
Remove Image.....	16
Conclusion	17

Introduction

Le but de ce TP est de gérer une galerie d'image. Les images et les albums seront stockés dans une base de données.

Partie I : Redis Data-store

Prise en main

Question 01

```
leo@leo:~/ESIR3/Cloud/TP4/td4-LeoGui/td4$ docker run -it --link redis-datastore:
redis-cli --rm redis redis-cli -h redis-datastore
redis-datastore:6379> set mykey testTD3
OK
redis-datastore:6379> get mykey
"testTD3"
redis-datastore:6379> set cle1 testTD1
OK
redis-datastore:6379> get cle1
"testTD1"
redis-datastore:6379> set counter 100
OK
redis-datastore:6379> incr counter
(integer) 101
redis-datastore:6379> incrby counter 50
(integer) 151
redis-datastore:6379> del mykey
(integer) 1
redis-datastore:6379> get mykey
(nil)
redis-datastore:6379>
```

Figure 1 : Test de redis

Comme on peut le voir ci-dessus, j'ai effectué quelques commandes de base de redis.

Question 02

Il faut faire un volume pour conserver les données. On crée dans un premier temps un dossier data où l'on va stocker les données puis on exécute la commande suivante.

```
leo@leo:~/ESIR3/Cloud/TP4/td4-LeoGui/td4/gallery/myappgallery$ docker run --name redis-datastore -v data:/data -d redis --appendonly yes
8bf5e4080f6e0db5d4f2535b698342eb03a8fb4590b209046d4beeade2166017
leo@leo:~/ESIR3/Cloud/TP4/td4-LeoGui/td4/gallery/myappgallery$ docker run --name redis-datastore -v data:/data -d redis --appendonly yes
docker: Error response from daemon: Conflict. The container name "/redis-datastore" is already in use by container "8bf5e4080f6e0db5d4f2535b698342eb03a8fb4590b209046d4beeade2166017". You have to remove (or rename) that container to be able to reuse that name.
See 'docker run --help'.
leo@leo:~/ESIR3/Cloud/TP4/td4-LeoGui/td4/gallery/myappgallery$ docker run -it --link redis-datastore:redis-cli --rm redis redis-cli -h redis-datastore
redis-datastore:6379> set key
(error) ERR wrong number of arguments for 'set' command
redis-datastore:6379> set key test
OK
redis-datastore:6379> get key
"test"
redis-datastore:6379> exit
leo@leo:~/ESIR3/Cloud/TP4/td4-LeoGui/td4/gallery/myappgallery$ docker rm -f redis-datastore
leo@leo:~/ESIR3/Cloud/TP4/td4-LeoGui/td4/gallery/myappgallery$ docker run --name redis-datastore -v data:/data -d redis --appendonly yes
6a71acbe3f64d3d1a52400958e6565b3ef8cd3c72977d0b4ea7af98ee6198797
leo@leo:~/ESIR3/Cloud/TP4/td4-LeoGui/td4/gallery/myappgallery$ docker run -it --link redis-datastore:redis-cli --rm redis redis-cli -h redis-datastore
redis-datastore:6379> get key
"test"
redis-datastore:6379> exit
```

Figure 2 : docker volume

Partie II : Architecture

Création d'une infrastructure Docker Node + Redis

Question 01

```
1 version: '2'
2 services:
3   web:
4     build: .
5     ports:
6       - "3000:3000"
7     volumes:
8       - './public/images:/usr/src/myappgallery/public/images'
9     links:
10      - redis
11   redis:
12     image: redis
13     ports:
14       - "6379:6379"
15     command:
16       - "--appendonly yes"
17     volumes:
18       - './data:/data'
19
```

Figure 3 : docker-compose.yml

```
1 FROM node:8
2
3 # Create app directory
4 WORKDIR /usr/src/myappgallery
5
6 # Install app dependencies
7 COPY package*.json ./
8
9 VOLUME /usr/src/myappgallery/public/images
10
11 RUN npm install
12
13 # Bundle app source
14 COPY . .
15
16 CMD [ "npm", "start" ]
17
```

Figure 4 : Dockerfile

Après avoir effectué la commande “docker-compose up --build” on peut voir qu’un fichier a été créé dans le dossier data. Ce fichier “appendonly.aof” montre bien que le volume a été effectué correctement et que les données sont persistantes.

Import des données

Question 01 et 02

Base de données album :

Nous allons utiliser « hmset » en suivant ce format :

Album:id nameAlbum "nameAlbumValue" thumbnail "thumbnailValue" idAlbum
"idAlbumValue"

```
hmset Album:1 nameAlbum cat thumbnail cat.png idAlbum 1
```

Figure 5 : Exemple table Album

On peut voir ci-dessus un exemple d'ajout via redis-cli.

Le champ « thumbnail » permet de récupérer la source d'une image pour pouvoir afficher une miniature. Cela permet donc à l'utilisateur de choisir la miniature pour son album. Les miniatures sont stockées dans le dossier "public/images/thumbnail".

```
function addAlbumInDB(nameAlbumValue,thumbnailValue){  
  
  //incrementation of albumID and add album in table  
  client.incr('albumIdCounter', function(err, id) {  
    client.hmset(["Album:" + id, "nameAlbum", nameAlbumValue , "thumbnail", thumbnailValue, "albumId", id]);  
  });  
  
  client.incr('albumCounter', function(err, id) {  
    });  
}
```

Figure 6 : Ajout d'un album dans la base de données

Ci-dessus le code JS permettant d'ajouter via l'UI un album.

Le champ id est rempli automatiquement. À chaque fois que l'on ajoute un album, on incrémente deux compteurs. Le premier "albumIdCounter" sert pour gérer les ID des albums, cela permet d'éviter d'avoir deux albums avec le même ID.

Le second "albumCounter" sert à compter le nombre d'album encore présent dans la table. Si on supprime un album, ce compteur est "décrémenté", tandis que l'autre compteur ne subit aucune modification.

Base de données image :

Nous allons utiliser « hmset » :

Image:id albumId "albumIdValue" imageName "imageNameValue" src "srcValue" imageId id

```
hmset Image:1 albumId cat imageName "Cute cate" src cat.png imageId id
```

Figure 7 : Exemple table Image

On peut voir ci-dessus un exemple d'ajout via redis-cli.

Le champ « albumId » permet d'associer l'image à un album. Stocker l'idAlbum permet de changer le nom de l'album sans qu'il y ait un impact sur les images. Les images sont stockées dans le dossier "public/images"

Ci-dessous le code JS permettant d'ajouter via l'UI un album.

```
function addImageInDB(albumIdValue,imageNameValue,srcValue){
  //incrementation of albumID and add album in table
  client.incr('imageIdCounter', function(err, id) {
    client.hmset(["Image:" + id, "albumId", albumIdValue , "imageName", imageNameValue, "src", srcValue, "imageId", id]);
  });

  client.incr('imageCounter', function(err, id) {
  });
}
```

Figure 8 : Ajout d'une image dans la base de données

Les deux compteurs fonctionnent de la même façon que pour les albums.

Question 03 et 04

Toutes les fonctions seront effectuées dans la fonction "app.get()" pour actualiser la page à chaque fois que l'on arrive dessus.

```
===== Get the number of album in DataBase =====/
var promiseCountNbAlbum = new Promise(function(resolve, reject){
  client.get('albumCounter',function(err,result){
    countNbAlbum = result;
    resolve(countNbAlbum);
  });
});

/===== Get the Id of the last album in DataBase =====/
promiseCountNbAlbum.then(function(value){
  var promiseCountIdNbAlbum = new Promise(function(resolve, reject){
    client.get('albumIdCounter',function(err,result){
      if(value == 0){
        res.render('index', {tabAlbum : tabAlbum})
      }
      countIdNbAlbum = result;
      resolve(countIdNbAlbum);
    });
  });
});

/===== Get infos about all albums in DataBase =====/
promiseCountIdNbAlbum.then(function(value){
  for(i = 1; i<= value ; i++){
    getAllAlbum(tabAlbum).then(function(value){
      res.render('index', {tabAlbum : tabAlbum})
    });
  };
});
});
```

Figure 9 : Récupérer les informations des albums

```
function getAllAlbum(tabAlbum){
  return new Promise(function(resolve, reject){
    client.hgetall('Album:' + i,function(err,result){
      if(result != null){
        var nameAlbum = result.nameAlbum;
        var thumbnail = result.thumbnail;
        var albumId = result.albumId;
        var albumDetails = {nameAlbum : nameAlbum, thumbnail : thumbnail, albumId : albumId}
        tabAlbum.push(albumDetails);

        //When we have all the info
        if(tabAlbum.length == countNbAlbum){
          resolve(tabAlbum);
        }
      }
    })
  })
};
```

Figure 10 : Récupérer les informations des albums

Pour récupérer les informations d'un album, on va utiliser des "Promise".

Ces fonctions permettent d'attendre une réponse avant d'effectuer la requête suivante. Les fonctions de Redis telles que "hget" ou "hgetall" ne renvoient pas un résultat direct, il faut donc attendre leur réponse.

Dans un premier temps, on récupère le nombre d'album présent dans la base de données (valeur de "albumCounter"). Ensuite, une fois qu'on a récupéré cette donnée, on récupère l'Id du dernier album ajouté (valeur de "albumIdCounter"). Une fois que l'on a terminé cette requête on effectue une boucle for pour parcourir tous les albums. En partant de l'ID 1 jusqu'au dernier ID (que l'on vient de récupérer).

Ensuite, dès que l'on récupère une valeur pour un album, on peut l'afficher. Ici, on récupère sa miniature, son nom et son ID. Ces valeurs sont ensuite stockées dans un tableau pour les afficher après.

Pour terminer cette "Promise", il faut que la taille du tableau de résultat soit égale au nombre d'album présent dans la base de données.

Une fois cette condition atteinte, on peut effectuer l'affichage de la page avec "res.render(.....)".

Pour récupérer les informations sur les images, il faut faire ceci :

```

/***** Get the number of Image in DataBase *****/
var promiseCountNbImage = new Promise(function(resolve, reject){
  client.get('imageCounter',function(err,result){
    countNbImage = result;
    resolve(countNbImage);
  });
});

/***** Get the Id of the last Image in DataBase *****/
promiseCountNbImage.then(function(value){
  var promiseCountIdNbImage = new Promise(function(resolve, reject){
    client.get('imageIdCounter',function(err,result){
      if(value == 0){
        res.render('images', {srcTable : srcTable , nbImagesInAlbum : nbImagesInAlbum, nameAlbum : nameAlbum, imageNameTable : imageNameTable})
      }
      nbImagesIDInAlbum = result;
      resolve(nbImagesIDInAlbum);
    });
  });
});

/***** Get infos about all Image in DataBase *****/
promiseCountIdNbImage.then(function(value){
  for(i = 1; i<= value ; i++){
    getAllImages(tabAllImages,tabImage,value,albumId,imageNameTable,srcTable,nbImagesInAlbum,countNbImage).then(function(value){
      nbImagesInAlbum = value.length;
      res.render('images', {srcTable : srcTable , nbImagesInAlbum : nbImagesInAlbum, nameAlbum : nameAlbum, imageNameTable : imageNameTable})
    });
  };
});
});
});

```

Figure 11 : Informations sur les images

```

function getAllImages(tabAllImages,tabImage,value,albumId,imageNameTable,srcTable,nbImagesInAlbum,countNbImage){
  return new Promise(function(resolve, reject){
    client.hgetall('Image:' + i,function(err,result){

      if(result != null){
        if(result.albumId == albumId){
          var imageName = result.imageName;
          imageNameTable.push(imageName);
          var srcImage = result.src;
          srcTable.push(srcImage);
          tabImage.push("image");
        }
      }

      tabAllImages.push("");

      //When we have all the info
      if(tabAllImages.length == countNbImage){
        resolve(tabImage);
      }
    });
  });
}

```

Figure 12 : Informations sur les images

Idem que pour récupérer les informations sur les albums, il faut effectuer des « Promise ». On compare l’ID de l’album que l’on a récupéré précédemment via les paramètres de l’URL avec l’albumID de chaque image. Si ces derniers correspondent alors on les ajoute dans un tableau. On récupère ainsi les informations telles que le nom de l’image et sa source.

La longueur du tableau “tabImage” me permet de savoir combien d’images se trouvent dans un album. Cela me sert lors de l’affichage de mon carrousel par la suite.

Idem que pour les albums, on récupère la longueur du tableau “tabAllImages” pour pouvoir faire une condition d'arrêt. Puis on affiche les informations.

Portage du code

Après avoir porté le code du TD 3 et effectué une partie des tâches demandées. En tapant l'adresse : <http://localhost:3000/> dans la barre de recherches, on obtient ceci :

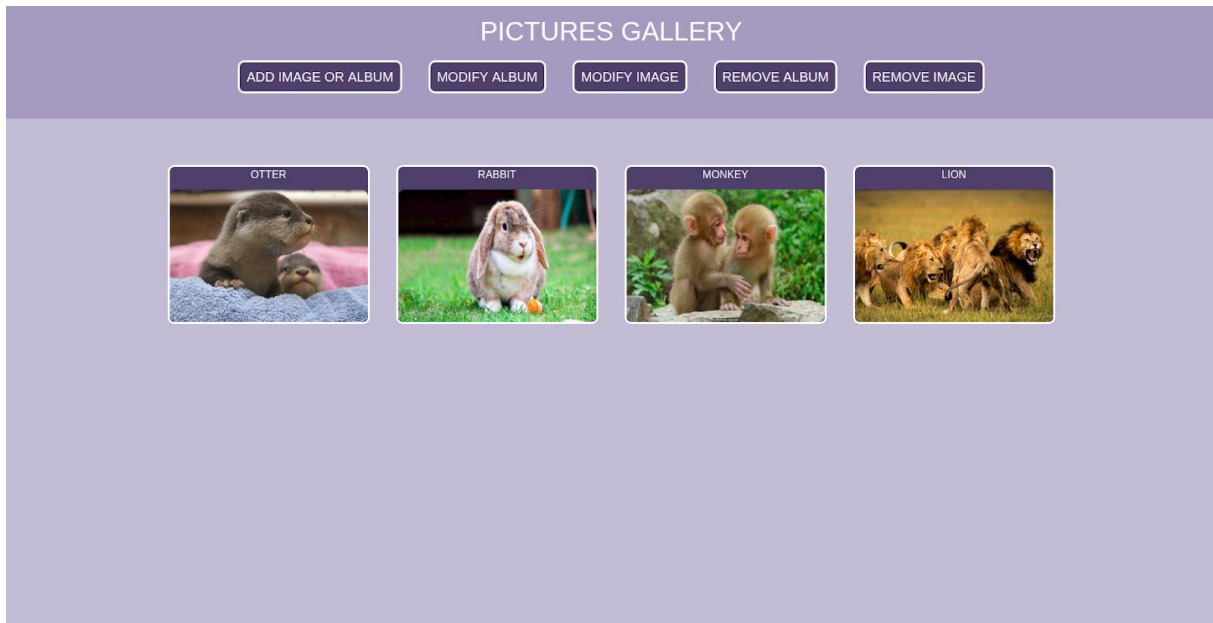


Figure 13 : Homepage

Sur cette page on peut voir la liste des albums ainsi que leur miniature. Cet affichage est dynamique et varie en fonction de l'ajout et de la suppression des albums.

Lorsque l'on clique sur l'album « monkey » contenant des photos on obtient ceci :

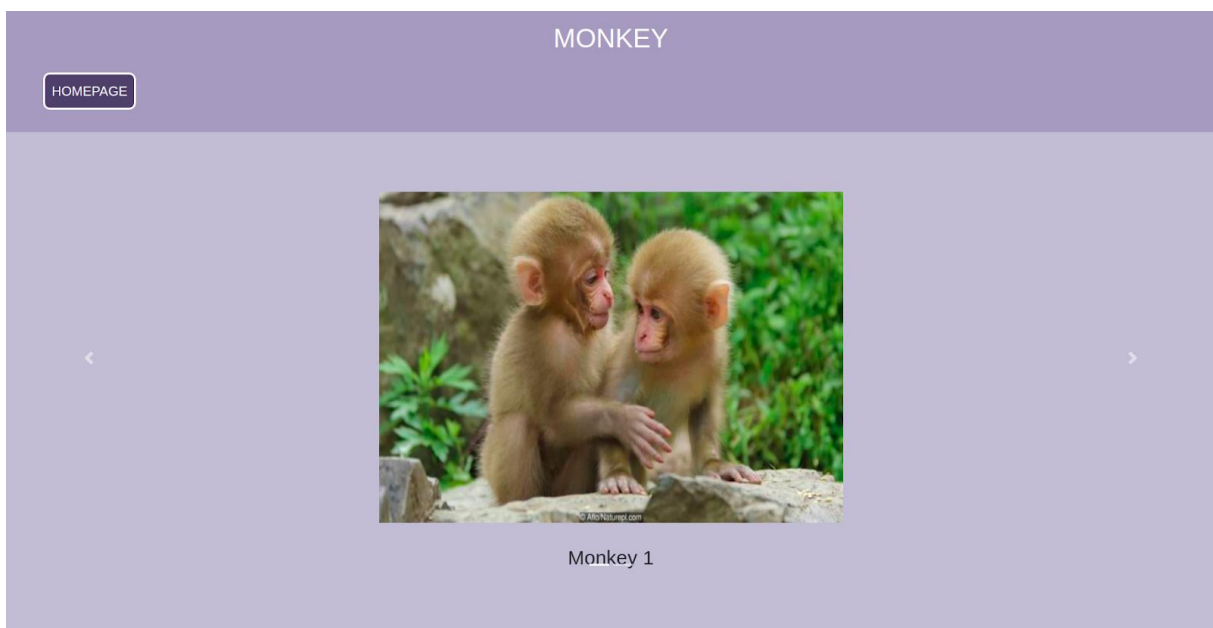


Figure 14 : Carrousel

Le lien du carrousel est : <http://localhost:3000/images/monkey/4>

On peut voir le titre de l'album en haut de l'écran ainsi qu'un carrousel. Ici, deux photos sont présentes dans l'album. Le titre de chaque photo est affiché juste en dessous.

Un bouton "homepage" permet de retourner sur la page d'accueil de notre site.

Si l'on clique sur l'album "otter" ne contenant pas de photos, on obtient ceci :

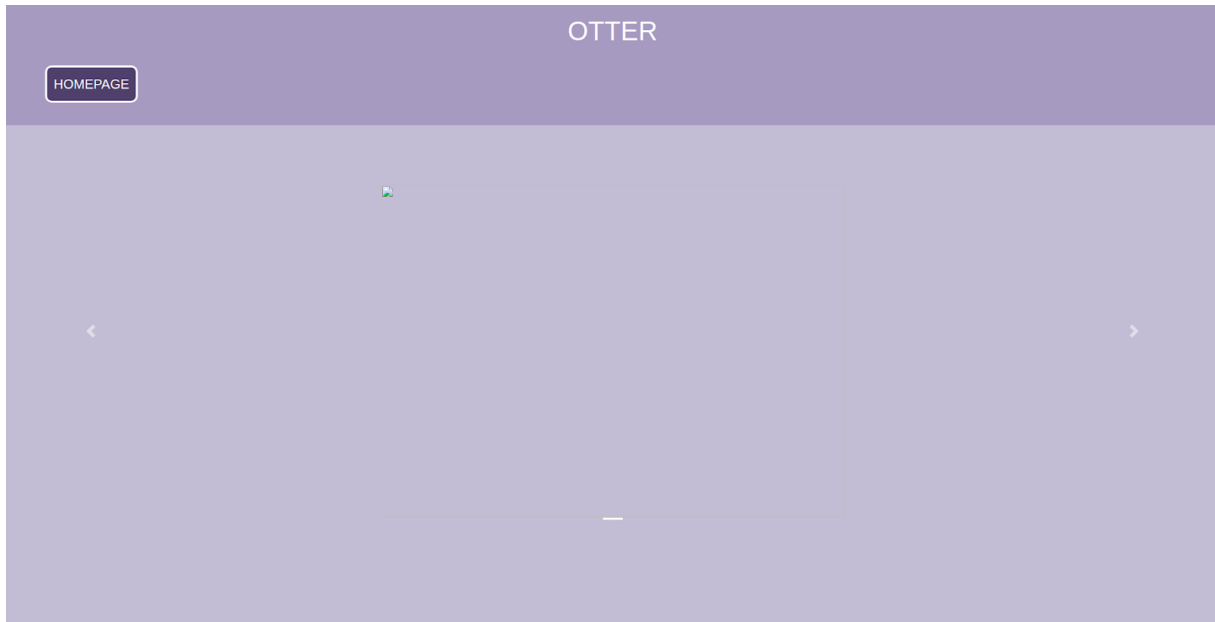


Figure 15 : Album Otter

Aucune photo n'a été trouvée dans l'album "otter" donc l'affichage est vide.

Sur la page d'accueil on peut également trouver 5 boutons : **"Add Image or Album"**, **"Modify Album"**, **"Modify Picture"**, **"Remove Album"**, **"Remove Image"**.

Add Image or Album

Lorsque l'on clique sur ce bouton, on arrive sur la page <http://localhost:3000/add/> :

ADD A NEW ALBUM OR A NEW IMAGE IN YOUR GALLERY

HOME PAGE REFRESH

ALBUM

Choisir un fichier Aucun fichier choisi

Album Name

submit

IMAGE

Choisir un fichier Aucun fichier choisi

Id Album

Image Name

submit

- 1 : otter
- 3 : rabbit
- 4 : monkey
- 5 : lion

Figure 16 : Ajout d'albums et d'images

Sur la droite, on peut voir tous les albums présents dans la base de données. L'ID est associés à leur nom pour faciliter l'ajout de l'utilisateur. Comme vous pouvez le remarquer, un album possédant l'ID numéro 2 a été supprimé précédemment. Cet ID ne sera donc plus jamais attribué.

Deux choix s'offrent ainsi à l'utilisateur. Soit il ajoute un album et dans ce cas, il doit choisir une miniature ainsi qu'un nom. Une fois l'ajout fait, pour pouvoir le voir dans la liste de droite, il faut qu'il clique sur le bouton "refresh" en haut de la page.

Il peut ensuite ajouter des images dans la base en entrant l'ID de l'album dans lequel il veut que l'image se trouve, le nom de l'image et l'image qu'il souhaite ajouter.

Pour pouvoir uploader des fichiers sur le serveur j’ai utilisé “multer js”.

```

/===== Store Images =====/

var storageImage = multer.diskStorage({
  {
    destination: 'public/images',
    filename: function ( req, file, cb ) {
      cb( null, file.originalname );
    }
  }
});
var uploadImage = multer( { storage: storageImage } );

router.post('/image', uploadImage.single('imageUploadFile'), (req, res) => {

  if (!req.file) {
    console.log("No file received");
  } else {
    console.log('file received');
    var srcValue = req.file.originalname;
    var imageNameValue = req.body.imageName;
    var albumIdValue = req.body.albumId;
    addImageInDB(albumIdValue, imageNameValue, srcValue);
  }
  res.render('add', {tabAlbum : tabAlbum});
});

function addImageInDB(albumIdValue, imageNameValue, srcValue){
  //incrementation of albumID and add album in table
  client.incr('imageIdCounter', function(err, id) {
    client.hmsset(["Image:" + id, "albumId", albumIdValue , "imageName", imageNameValue, "src", srcValue, "imageId", id]);
  });

  client.incr('imageCounter', function(err, id) {
  });
}

```

Figure 17 : Upload multer

Les images sont stockées dans le dossier “public/images”. Via le formulaire, on récupère le nom du fichier qui est ici “imageUploadFile” puis multer s’occupe d’upload le fichier. Les images sont stockées dans le dossier « public/images » tandis que les miniatures sont stockées dans le dossier « public/images/thumbnail ».

```

form(action="/add/image" id="form_image" method="post" enctype="multipart/form-data")
  input(type="file" name="imageUploadFile" value="fileupload" id="fileupload")
  input(class="form-control" name="albumId" type="text" placeholder="Id Album")
  input(class="form-control" name="imageName" type="text" placeholder="Image Name")
  input(class="btn btn-primary" form="form_image" type="submit" value="submit")

```

Figure 18 : Formulaire d'ajout

On récupère ensuite les autres valeurs du formulaire pour les stocker dans la base de données.

Modify Album

Lorsque l'on clique sur ce bouton, on arrive sur la page <http://localhost:3000/modifyAlbum/> :

MODIFY ALBUM IN YOUR GALLERY

HOMEPAGE REFRESH

ALBUM

ID Album

Name Album

submit

- 1 : otter
- 3 : rabbit
- 4 : monkey

Figure 19 : Modification d'album

L'utilisateur peut modifier le nom d'un album. Pour faire cela, il lui suffit d'entrer l'ID de l'album dans le formulaire puis d'ajouter un nouveau nom d'album. Ensuite lorsqu'il rafraichit la page, le nom a changé dans la colonne de droite mais également dans la base de données. Cela implique également un changement sur la page d'accueil. Par exemple, si je veux modifier l'album « rabbit » et que son nom soit désormais "bunny". Alors je rentre l'ID numéro 3 puis le nom "Bunny". Après rafraichissement, on obtient ceci :

MODIFY ALBUM IN YOUR GALLERY

HOMEPAGE REFRESH

ALBUM

ID Album

Name Album

submit

- 1 : otter
- 3 : bunny
- 4 : monkey

Figure 20 : Modification de l'album "rabbit"

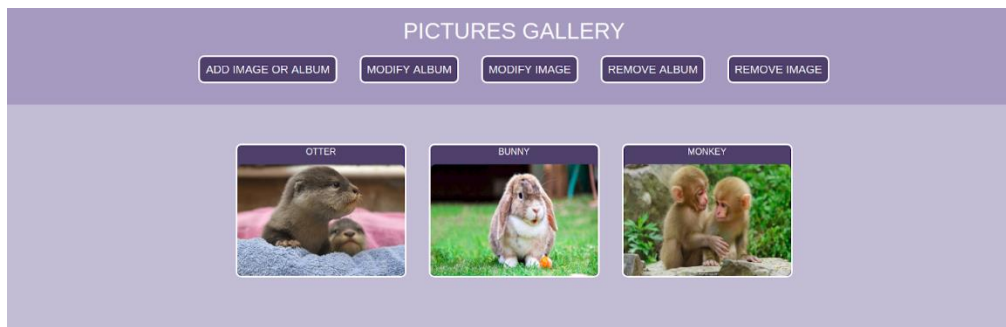


Figure 21 : Page d'accueil après modification

Le nom a bien été changé partout. À noter que l'album de "lion" a été supprimé auparavant c'est pour cela qu'il n'apparaît plus sur la page d'accueil.

Pour faire cette modification dans la base de données, il suffit, après avoir fait une recherche de tous les albums, de modifier juste le nom dans "hmset"

```
getAlbumInfo(albumId).then(function(value){
  client.hmset(["Album:" + albumId, "nameAlbum", nameAlbum, "thumbnail", value, "albumId", albumId]);
});
```

Figure 22 : Modification du nom de l'album

Modify Image

Lorsque l'on clique sur ce bouton, on arrive sur la page <http://localhost:3000/modifyImage> :

Le fonctionnement est le même que pour la modification d'album. Sur la droite, on peut voir l'id de l'image avec sa source mais également le nom associé à cet ID.

Remove Album

Lorsque l'on clique sur ce bouton, on arrive sur la page <http://localhost:3000/removeAlbum/>

REMOVE ALBUM IN YOUR GALLERY

HOMEPAGE REFRESH

ALBUM

Warning ! If you delete an album, you will delete all the images in that album !

ID Album

submit

- 1 : otter
- 3 : rabbit
- 4 : monkey
- 5 : lion

Figure 23 : Suppression d'album

Pour l'affichage, le fonctionnement est le même que pour l'ajout d'album et d'image.

L'utilisateur entre l'ID de l'album qu'il souhaite supprimer. Par exemple, s'il veut supprimer l'album "lion", il doit mettre l'ID numéro 5. Ensuite après un rafraichissement, on obtient ceci :

REMOVE ALBUM IN YOUR GALLERY

HOMEPAGE REFRESH

ALBUM

Warning ! If you delete an album, you will delete all the images in that album !

ID Album

submit

- 1 : otter
- 3 : rabbit
- 4 : monkey

Figure 24 : Suppression de l'album "lion"

L'album « lion » n'est plus dans la liste à droite donc il a bien été supprimé. Il n'apparaîtra plus sur la page principale. Il faut savoir que lorsque l'on supprime un album, toutes les photos se trouvant dans ce dernier sont supprimées.

```

/===== Remove thumbnail =====/

function RemoveFile(srcImage){
  fs.stat('public/images/thumbnail/' + srcImage, function (err, stats) {

    if (err) {
      return console.error(err);
    }

    fs.unlink('public/images/thumbnail/' + srcImage,function(err){
      if(err){return console.log(err);}
      console.log('file deleted successfully');
    });
  });
};

function RemoveImage(albumId){

  countNbAlbum = 0;

  /===== Get the number of album in DataBase =====/
  var promiseCountNbAlbum = new Promise(function(resolve, reject){
    client.get('albumCounter',function(err,result){
      countNbAlbum = result;
      resolve(countNbAlbum);
    });
  });

  /===== Get the Id of the last album in DataBase =====/
  promiseCountNbAlbum.then(function(value){
    var promiseCountIdNbAlbum = new Promise(function(resolve, reject){
      client.get('albumIdCounter',function(err,result){
        countIdNbAlbum = result;
        resolve(countIdNbAlbum);
      });
    });

    /===== Get infos about all albums in DataBase =====/
    promiseCountIdNbAlbum.then(function(value){
      for(i = 1; i<= value ; i++){
        getSrcImage(albumId).then(function(value){
          RemoveFile(value);
          RemoveImageInAlbum(albumId);
          RemoveAlbumInDB(albumId);
        });
      }
    });
  });
};

```

Figure 25 : Suppression d'album

Dans un premier temps, on fait une recherche de tous les albums, puis une fois que l'on a récupéré l'ID de l'album que l'on souhaite supprimer, on peut récupérer toutes ses informations, notamment la source de la miniature.

Une fois cette source récupérée, on effectue la fonction RemoveFile() qui prend en paramètre la source de la miniature. Cette fonction va aller chercher dans le dossier "public/images/thumbnail" puis va supprimer l'image correspondante.

Ensuite on effectue la fonction `RemoveImageInAlbum()`. Cette fonction permet de parcourir toutes les images et de stocker dans un tableau toutes les informations sur les images correspondant à l'albumID. Ensuite on fait la même chose que pour la miniature, on supprime toutes les images correspondant aux sources relevées précédemment dans le fichier "public/images/".

Enfin, on supprime l'album dans la base de données.

```
function RemoveAlbumInDB(albumIdValue){
  var key = "Album:" + albumIdValue;
  client.del(key);
  client.decr('albumCounter', function(err, id) {
  });
};
```

Figure 26 : Suppression d'un album dans la base de données

Grâce à l'albumId récupérer précédemment, on supprime l'album dans la base de données.

On "décrémente" le compteur du nombre d'album présent dans la base de données.

Remove Image

Lorsque l'on clique sur ce bouton, on arrive sur la page <http://localhost:3000/removeImage/>

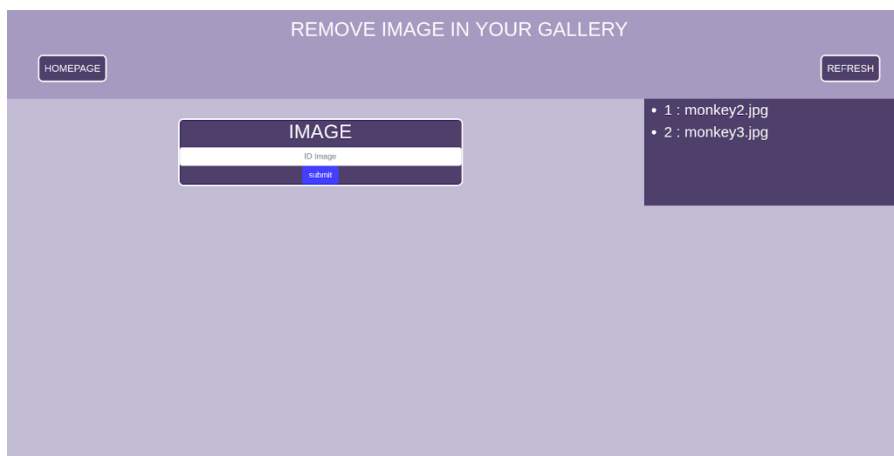
The screenshot shows a web application interface with a purple header and a light purple body. The header contains the title "REMOVE IMAGE IN YOUR GALLERY" in the center, a "HOMEPAGE" button on the left, and a "REFRESH" button on the right. The main content area is divided into two sections. On the left, there is a form titled "IMAGE" with a label "ID Image:" and a text input field. Below the input field is a blue "submit" button. On the right, there is a dark purple box containing a list of image IDs: "1 : monkey2.jpg" and "2 : monkey3.jpg".

Figure 27 : Remove image

Pour l'affichage, le fonctionnement est le même que pour l'ajout d'album et d'image. La différence est que sur la droite on ne voit plus les albums mais les images.

Le fonctionnement pour supprimer une image est le même que pour supprimer un album. Il suffit de récupérer son ID puis de le supprimer dans la base de données.

Conclusion

Dans ce TP, j'ai pu faire une ajouter et supprimer des fichiers de façon dynamique. Cependant, par manque de temps, je n'ai pas réussi à implémenter la gestion des utilisateurs.