NodeJS : Web Application

- Express Generator
- **Upload de fichiers**
- Modules NPM pour l’upload
- Formidable
- Application pratique

Express Generator

- **Nouveau projet** avec **Express Generator**
- Pas besoin de l’installer de nouveau si global
  - **express uploadFichier --view=ejs**
  - **npm install**
  - **set DEBUG=uploadFichier:* && npm start**
- L'arborescence des fichiers va être créée et le moteur de template choisi est **EJS**
- Les modules sont installés
- Démarrage en mode DEBUG

Installation de l'application

```
PS E:\nodeProjects\demo_upload> express uploadFichier --view=ejs

   create : uploadFichier\
   create : uploadFichier\public\
   create : uploadFichier\public\javascripts\
   create : uploadFichier\public\images\
   create : uploadFichier\public\stylesheets\
   create : uploadFichier\public\stylesheets\style.css
   create : uploadFichier\routes\
   create : uploadFichier\routes\index.js
   create : uploadFichier\routes\users.js
   create : uploadFichier\views\
   create : uploadFichier\views\error.ejs
   create : uploadFichier\views\index.ejs
   create : uploadFichier\app.js
   create : uploadFichier\package.json
   create : uploadFichier\bin\
   create : uploadFichier\bin\www

   change directory:
     > cd uploadFichier

   install dependencies:
     > npm install

   run the app:
     > SET DEBUG=uploadfichier:* & npm start
```

Modification de l’application

- Dans ***package.json***, ajouter dans "scripts"
  - **"devstart" : "nodemon ./bin/www"**
- Et redémarrer l'application avec
  - **npm run devstart**
- Pas de gestion des utilisateurs
  - Commenter les lignes 8 et 23 dans **app.js**
  - (Supprimer le fichier **routes/users.js**)
- Page d’accueil : **lien** vers un formulaire d’upload
  - Modifier la vue dans **views/index.ejs**

Upload de fichiers

- **Upload** = transfert de fichiers de l'ordinateur d'un utilisateur vers le serveur web (c'est l'inverse du download ou téléchargement) pour mettre en ligne des photos, des images,…
- Il faut un **formulaire**, avec quelques particularités :
  - la **method="post"** pour gros transfert de données
  - l'option **enctype="multipart/form-data"** indiquant un échange de données
  - un **champ** input de **type=file** pour sélectionner le fichier dans le système de fichiers de son PC
  - Un **bouton** d'envoi **type=submit**

Comment faire avec Express

- Dans une application NodeJS et Express, plusieurs **modules NPM** sont disponibles pour faire un **upload** de fichiers
- **Les plus connus** : Busboy, Express-fileupload, Formidable, Multer, Multiparty

  voir le comparatif sur npmcompare.com
- La plupart des tutos sur Internet parlent de **Formidable** et **Multer :** OpenClassRooms , TheServerSide.com , StackAbuse.com , twilio.com , GeeksForGeeks.org , … + Youtube

Formidable

- Attention : il y a **plusieurs versions** ; ici, on va utiliser Formidable v2
- **Installation** :
  npm install formidable@v2

  ou bien
  npm install formidable@v3

```
{
  "name": "uploadfichier",
  "version": "0.0.0",
  "private": true,
  ▷ (débogage)
  "scripts": {
    "start": "node ./bin/www",
    "devstart": "nodemon ./bin/www"
  },
  "dependencies": {
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "ejs": "~2.6.1",
    "express": "~4.16.1",
    "formidable": "^2.0.1",
    "http-errors": "~1.6.3",
    "morgan": "~1.9.1"
  },
  "devDependencies": {
    "nodemon": "^2.0.15"
  }
}
```

Formulaire

- Dans la page d'accueil (view : index.ejs), on crée un **formulaire** d'upload

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <form action='/' method="POST" enctype='multipart/form-data'>
      <label>Nom du fichier :</label><input type="file" name="upload" multiple="" />
      <br><br>
      <input type="submit" value="Envoyer le fichier" />
    </form>
  </body>
</html>
```

Dossier uploads

- Dans le dossier public (par exemple), on crée un **nouveau répertoire** qui stockera les **fichiers uploadés**

```
PS E:\nodeProjects\demo_upload\uploadFichier> cd public
PS E:\nodeProjects\demo_upload\uploadFichier\public> mkdir uploads


    Répertoire : E:\nodeProjects\demo_upload\uploadFichier\public


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         15-03-21     15:32                uploads
```

Pages de résultats

- On peut aussi créer deux autres pages :
  - Une page pour confirmer que **l'upload s'est bien déroulé** (avec, par exemple, les informations du transfert)

    => **uploaded.ejs**
  - Une page pour signaler une **erreur**

    => **error.ejs**

Upload réussi

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Le fichier <%= nom %> a été uploadé avec succès !</p>
    <p>Sa taille est de <%= taille %> octets</p>
    <p>Son type MIME est : <%= type %></p>
    <hr />
    <p><a href="./">Uploader un autre fichier</a></p>
  </body>
</html>
```

Upload - erreur

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= message %></h1>
    <h2><%= error %></h2>
    <hr>
    <p><a href="./">Uploader un autre fichier</a></p>
  </body>
</html>
```

Module formidable

- Dans le routeur principal (routes/index.js), on **ajoute le module formidable** avec require
- L'accès à la page n'est pas modifié

```
var express = require('express');
var router = express.Router();

var formidable = require('formidable');

/* GET home page. */
router.get('/', function(req, res, next) {
  res.render('index', { title: 'Express' });
});
```

Paramétrage de l’envoi

- Après avoir créé un objet **form** pour recevoir un nouvel envoi de données, on définit ses différents **paramètres** (taille maximale des fichiers, types autorisés,…) avec les options disponibles

```
/* Envoi du formulaire d'upload */
router.post('/', function(req,res){
  /* nouveau formulaire entrant à traiter */
  var form = new formidable.IncomingForm();
  /* définir une taille maximum pour les fichiers à uploader */
  form.maxFileSize = 2 * 1024 * 1024;  // 2Mo
  /* analyser le formulaire */
  form.parse(req);

  /* Filtrer sur les types de données acceptées */
  fileTypes = ['image/jpeg', 'image/png', 'image/gif'];
```

Vérification du contenu

- On vérifie que les différentes **parties reçues** sont **conformes** aux paramètres (onPart), soit on les traite, soit on lève une erreur

```
form.onPart = function(part) {
    if (fileTypes.indexOf(part.mimetype) === -1) {
        // Here is the invalid file types will be handled.
        // You can listen on 'error' event
        form._error(new Error('File type is not supported : ' + part.mimetype));
    }
    if (!part.originalFilename || fileTypes.indexOf(part.mimetype) !== -1) {
        // Let formidable handle the non file-pars and valid file types
        form._handlePart(part);
    }
};
```

Début de l'upload

- On détecte que l'upload est commencé quand l'événement **fileBegin** est levé. Les données concernant le fichier sélectionné peuvent être récupérées via l'objet **file**

```
/* Quand l'événement fileBegin est levé, c'est le début de l'upload */
form.on('fileBegin', function(name, file){
  /* on uploade le fichier et on affiche en console son nom */
  console.log("Fichier uploadé : " + file.originalFilename);
  /* on sauvegarde le fichier uploadé dans le répertoire 'uploads' */
  file.filepath = __dirname + '/../public/uploads/' + file.originalFilename;
  console.log(file.filepath);
});
```

Upload terminé

- Quand le transfert est **terminé avec succès**, un événement **file** est levé, on peut confirmer en envoyant les données sur la page

```
/* A la fin de l'upload, on se redirige vers la page /uploaded pour confirmer */
form.on('file', function(name,file){

  console.log("Nom original du fichier : " + file.originalFilename);
  console.log("Taille du fichier : " + file.size);
  console.log("Type de fichier : " + file.mimetype);

  res.render('uploaded', {title:'Upload', nom: file.originalFilename, taille: file.size,
  type: file.mimetype});
});

/* En cas de succès, on affiche dans la console OK */
form.on('end', function() {
  console.log('Upload OK !');
});
```

Upload raté

- Quand le transfert s'est **mal terminé** avec une erreur, un événement **error** est levé avec les indications des causes du problème.
  On peut l'afficher dans la page d'erreur.

```
/* En cas d'erreur, on se redirige vers la page /error */
form.on('error', function(err){
  console.log('Erreur : '+ err);
  console.log('Stack : '+ err.stack);

  res.render('error', {title:'Erreur', message: "Une erreur s'est produite", error: err});
});
```

Autres possibilités

- Il y a d’autres options disponibles et d’autres événements qui peuvent être gérés avec ce module formidable
- Avec les autres versions de ce module, ou avec d’autres modules (multer,…) les noms des propriétés, des méthodes et des événements peuvent changer.
- Il faudra toujours suivre la documentation pour s’adapter aux évolutions