EXERCICE - VIDÉO À LA DEMANDE

Pré-requis

Javascript ES6, NodeJs, MongoDB

Lire les parties : NodeJs, Modules NodeJs

Objectif de la formation

Renforcer les connaissances Javascript/NodeJs, MongoDB.
Utilisation d'une API (TheMovieDb).

Enoncé

Créer un site de VOD (sans la partie streaming ;-))

Côté back vous devez créer une partie administration permettant d'ajouter un film sur le site VOD (les informations devront être stockées en mongodb). Les données seront récupérées via l'API allociné, ce qui remplira automatiquement les champs du formulaire de création.

Côté front vous devez créer l'affichage des films disponible sur le site, l'utilisateur doit voir les affiches des films et pouvoir entrer dans la fiche du film pour voir le synopsis et les infos (année de sortie,réalisateur(s), acteurs, etc) stockées dans notre BDD mongoDB (l'API allociné ne sert que côté back).

En vous appuyant sur la documentation suivante :
https://developers.themoviedb.org/3
Vous pourrez facilement récupérer les informations sur les films.

Avant de commencer nous allons découvrir Express via un workshop

DÉCOUVERTE EXPRESS WORKSHOP

Commencez par découvrir express avec le workshop expressworks

```
npm install -g expressworks
```

En cas de problème d'exécution de script il faut démarrer un powershell en mode administrateur et taper la commande

```
set-executionpolicy unrestricted
```

Pour restreindre à nouveau l'exécution des scripts

```
set-executionpolicy remotesigned
```

Pour lancer les exercices expressworks tapez la commande :

```
expressworks
```

VIDÉO À LA DEMANDE

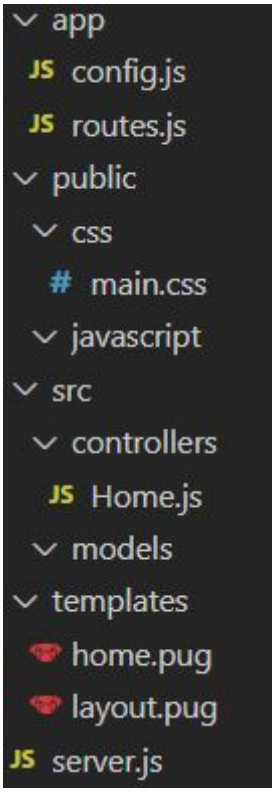
Commençons par créer notre package.json

```
npm init
```

Répondez à chaque question.

Architecture

Commencez par créer cette architecture :



Le fichier **app/config.js**, contiendra les informations de configuration (par exemple le port d'écoute de notre serveur HTTP, les accès à notre base de données, etc).

Le fichier **app/routes.js** contiendra l'ensemble des déclarations de nos routes.

Les fichiers dans le répertoire **src/controllers/** seront nos différents controllers (les méthodes des contrôleurs seront appelées depuis les routes), dans la majorité des cas les controllers (les méthodes) retourneront le rendu des templates.

le fichier **./server.js** et notre fichier de démarrage nous devrons démarrer notre serveur à partir de ce fichier.

Vous pouvez si vous le souhaitez utiliser stylus pour gérer votre CSS !

Mise en place du serveur HTTP

Pour créer notre serveur HTTP nous allons utiliser express, commençons par l'installer :

```
npm install express --save
```

Vous pouvez commencer à mettre en place votre serveur HTTP, pensez à préciser le dossier statique (**./public**)

Installez le moteur de template PUG :

```
npm install pug
```

Intégrez PUG à votre application express, précisez le chemin vers les templates.

Si vous souhaitez utilisez stylus pensez à l'installer également :

```
npm install stylus
```

Intégrez stylus à votre application express via le middleware.

```
8.  app.use(require('stylus').middleware(__dirname + '/public'));
```

Browser-Refresh

Nous allons mettre en place un outil très pratique Browser-refresh, celui-ci permet de relancer automatiquement votre serveur dès qu'il y aura une modification dans votre code et pourra également rafraîchir votre navigateur, cela est très pratique quand vous aurez à travailler sur le contenu html et/ou CSS de votre site !

Pour l'installer tapez la commande :

```
npm install -g browser-refresh --save-dev
```

Ainsi pour lancer votre serveur vous devez taper la commande :

```
browser-refresh server
```

et non plus

```
node server
```

Cependant nous allons modifier notre fichier **package.json** afin de faciliter le démarrage !
Modifiez la partie script de votre fichier **package.json :**

```
1. {
2.   "name": "vod",
3.   "version": "1.0.0",
4.   "description": "Projet vod",
5.   "main": "server.js",
6.   "scripts": {
7.     "test": "echo \"Error: no test specified\" && exit 1",
8.     "start": "browser-refresh server.js"
9.   },
10.  "author": "Cyril",
11.  "license": "ISC",
12.  "dependencies": {
13.    "express": "^4.17.1",
14.    "pug": "^2.0.4"
15.  },
16.  "devDependencies": {
17.    "browser-refresh": "^1.7.3"
18.  }
19. }
```

En modifiant la ligne 8 de notre package.json, nous avons changé

```
9.      "start": "node server.js"
```

en

```
8.      "start": "browser-refresh server.js"
```

Nous pouvons démarrer notre serveur avec la commande :

```
npm start
```

Codons notre Server.js

Exemple fichier **config.js**

```
1. module.exports = {
2.     port : 3000
3. };
```

Exemple fichier **routes.js**

```
1. module.exports = (app) => {
2.     app.get('/', (req, res) => {
3.         res.send("Hello World");
4.     });
5. };
```

Voici à quoi pourrait ressembler notre fichier **server.js**

```
1. const express = require('express');
2. const app = express();
3. const path = require('path');
4. const config = require('./app/config');
5.
6. //---------------------------------------------------------------------
7. //      Mise en place du moteur de template
8. //---------------------------------------------------------------------
9. app.set('views', path.join(__dirname, 'templates'));
10.app.set('view engine', 'pug');
11.
12.//---------------------------------------------------------------------
13.//      Mise en place du répertoire static
14.//---------------------------------------------------------------------
15.app.use(express.static(path.join(__dirname, 'public')));
16.
17.//---------------------------------------------------------------------
18.//      Chargement des routes
19.//---------------------------------------------------------------------
20.require('./app/routes')(app);
21.
22.//---------------------------------------------------------------------
23.//     Ecoute du serveur HTTP
24.//---------------------------------------------------------------------
25.app.listen(config.port,() => {
26.    console.log(`Le serveur est démarré : http://localhost:${config.port}` );
27.});
```

A cette étape si nous lançons notre commande “**npm start**”, nous devrions pouvoir voir un “**Hello world**” à l’adresse “**http://127.0.0.1:3000**”

⚠ Important

Quand vous exportez une classe avec module.exports, le nom de cette classe n’a réellement aucune importance, vous pourriez d’ailleurs utiliser une classe anonyme, cela ne changerait rien au comportement de votre code.

Utilisation des contrôleurs et des templates

Créons notre Contrôleur **src/controllers/Home.js**

```
module.exports = class Home {
    print(request, response) {
        response.render('home');
    }
};
```

Le contrôleur fait appel à notre fichier **templates/home.pug**

Voici une base (extrêmement basique) de notre fichier pug

```
div Hello world !!!
```

Cela crée une balise **div** contenant le texte "**Hello world !!!**"

Modifions le fichier **app/routes.js** pour qu'il fasse appel à notre contrôleur

```
module.exports = (app) => {
    app.get('/', (req, res) => {
        let Home = require('../src/controllers/Home.js')
        let Controller = new Home();
        Controller.print(req, res);
    });
};
```

A partir de la documentation de PUG :
https://pugjs.org/language/inheritance.html

Créez une structure HTML dans le fichier **templates/layout.pug**, et faire en sorte que **templates/home.pug** soit étendue de ce layout et créez votre page d'accueil.

Utilisez des fichiers CSS, javascript si nécessaire, vous pouvez également utiliser un framework CSS comme bootstrap, semantic,material design, etc ou des librairie JS comme jQuery.

Browser-Refresh (Hot-Reload)

Dans votre fichier **template/layout.pug** ajoutez votre head ou à la fin du body le code :

```
12.         script(src=process.env.BROWSER_REFRESH_URL)
```

Cela permettra à votre navigateur de recharger (ré-actualiser) tout seul à chaque modification de fichier.

Pour générer votre contenu PUG à partir du code HTML vous pouvez utiliser le site :
https://html2pug.now.sh/

Ajout du formulaire d'inscription

Nous allons ajouter le lien dans notre template de base (**templates/layout.pug**)

```
10.         header
11.             nav
12.                 a(href="/") Accueil
13.                 a(href="/inscription") Inscription
```

Actuellement la route /inscription (url dans notre application) n'existe pas nous devons la créer !

Dans le fichier **app/routes.js** on ajoute la route pour l'inscription (dans la fonction fléchée bien entendu)

```
10.     app.get('/inscription', (req, res) => {
11.         let UserRegister =
    require('../src/controllers/UserRegister.js')
12.         let Controller = new UserRegister();
13.         Controller.print(req, res);
14.     });
```

Cette route fait appel au controller UserRegister qui n'existe pas, nous allons donc le créer.

Créez le fichier **src/controllers/UserRegister.js**

```
module.exports = class UserRegister {
    print(request, response) {
        response.render('user/form_register');
    }
};
```

Ce contrôleur fait appel à un template qui n'existe pas, nous devons donc le créer.
Créez le fichier **templates/user/form_register.pug**

```
extends ../layout.pug

block content
    h1 Inscription

    form(method="post")
        .form-group
            label(for="firstname") Prénom
            input.form-control(type="text" id="firstname" name="firstname" placeholder="Votre prénom")
        .form-group
            label(for="lastname") Nom
            input.form-control(type="text" id="lastname" name="lastname" placeholder="Votre nom")
        .form-group
            label(for="email") Adresse E-mail
            input.form-control(type="email" id="email" name="email" placeholder="Votre adresse email")
        .form-group
            label(for="password")
            input.form-control(type="password" id="password" name="password" placeholder="Choisissez un mot de passe")

        .form-group
            button.btn.btn-primary(type="submit") S'inscrire
```

Prise en compte de l'inscription

Nous devons dans un premier temps récupérer les accès à notre base de données MongoDB : [MONGODB ATLAS]

Pour nous connecter à notre BDD MongoDB nous allons utiliser le module Mongoose : [https://mongoosejs.com/docs/]

Nous devons installer mongoose

```
npm install mongoose
```

Effectuons un test de connexion à la base de données avant de prendre en compte notre inscription, créez un fichier **./test_bdd.js** (par exemple) en le positionnant à la racine de votre projet.

```
1. const mongoose = require('mongoose');
2. mongoose.connect(
3.     'mongodb+srv://username:password@cluster.mongodb.net/vod',
4.     {connectTimeoutMS : 3000, socketTimeoutMS: 20000, useNewUrlParser:
   true, useUnifiedTopology: true }
5. );
6. const db = mongoose.connection;
7. db.once('open', () => {
8.    console.log(`connexion OK !`);
9. });
```

Pensez à modifier les valeurs en rouge (`username:password@cluster`) par vos données réelles. Pour tester votre connexion ouvrez un nouveau terminal et lancer la commande suivante (en vous assurant d'être dans le bon répertoire).

```
node test_bdd.js
```

Si vous voyez le message dans "Connexion OK !" dans le terminal c'est que vous avez réussi.
Il va falloir maintenant gérer cette connexion sur notre **server.js**

Dans votre fichier **server.js** ajoutez la connexion

```
25. //-----------------------------------------------------------------------
26. // Connexion à MongoDB
27. //-----------------------------------------------------------------------
28. const mongoose = require('mongoose')
29. mongoose.connect(
30.     config.mongodb,
31.     {connectTimeoutMS : 3000, socketTimeoutMS: 20000, useNewUrlParser:
    true, useUnifiedTopology: true }
32. )
33. mongoose.connection.once('open', () =>  {
34.     console.log(`Connexion au serveur MongoDB OK`)
35. })
```

Dans un premier temps nous avons besoin de créer le Schema,
le schéma est une entity, cela signifie que c'est une structure représentant les éléments que nous allons insérer en base de données, mais pour rappel en MongoDB la structure des éléments n'est pas obligatoirement fixe, nous pourrions avoir des attributs en plus ou en moins cela ne serait pas un problème pour mongoDB.

Nous devons ajouter l'attribut **mongodb** dans le fichier **app/config.js**

```
1. module.exports = {
2.     mongodb :
    'mongodb+srv://username:password@cluster.mongodb.net/ecommerce',
3.     port : 3000
4. }
```

Pensez à modifier les valeurs en rouge (username:password@cluster) par vos données réelles.

Dans notre cas nous ferons en sorte de renvoyer directement le model MongoDB, le model MongoDB est un objet mongoose permettant la manipulation d'une collection et qui attend en paramètre le nom de la collection et son schéma. Notre fichier **src/models/UserMongoDB.js** pourrait ressembler à ça :

```
1. const mongoose = require('mongoose');
2. const UserSchema = mongoose.Schema({
3.     firstname: { type: String, match:
   /^[a-zA-ZàáâäãåąčćęèéêëėįìíîïłńòóôöõøùúûüųūÿýżźñçčšžÀÁÂÄÃÅĄĆČĖĘÈÉÊËÌÍÎÏĮ
   ŁŃÒÓÔÖÕØÙÚÛÜŲŪŸÝŻŹÑßÇŒÆČŠŽ∂ð ,.'-]+$/ },
4.     lastname: { type: String, match:
   /^[a-zA-ZàáâäãåąčćęèéêëėįìíîïłńòóôöõøùúûüųūÿýżźñçčšžÀÁÂÄÃÅĄĆČĖĘÈÉÊËÌÍÎÏĮ
   ŁŃÒÓÔÖÕØÙÚÛÜŲŪŸÝŻŹÑßÇŒÆČŠŽ∂ð ,.'-]+$/ },
5.     email : {  type: String },
6.     password : { type: String },
7.     isAdmin: { type: Boolean, default: false },
8.     date: { type: Date, default: Date.now }
9. });
10. module.exports = mongoose.model('User', UserSchema);
```

Nous ferons appel à ce fichier depuis notre modèle principal (**src/models/User.js**), c'est ce modèle qui sera instancié dans le controller et c'est ce modèle qui possédera les différentes méthodes CRUD pour la manipulation de notre collection"users", il utilisera les méthodes mongoose via le modèle généré par mongoose qui est exporté depuis le fichier **src/models/UserMongoDB.js**

Voici une version basique de notre fichier **src/models/User.js**

```
11. const UserMongo = require('./UserMongoDB.js');
12.
13. module.exports = class User {
14.
15.     add(lastname, firstname, email, password) {
16.         return UserMongo.create({lastname, firstname, email,
   password});
17.     }
18. }
```

Nous devons créer une nouvelle route ! Cette route peut être la même que celle affichant le formulaire d'inscription à la différence qu'elle doit être en post. Dans le fichier **app/routes.js** on ajoute la route pour l'inscription (dans la fonction fléchée bien entendu)

```
19.     app.post('/inscription', (req, res) => {
20.         let UserRegister =
    require('../src/controllers/UserRegister.js');
21.         let Controller = new UserRegister();
22.         Controller.process(req, res);
23.     });
```

Nous allons devoir dans notre controller gérer les données soumises par le formulaire pour cela nous allons ajouter le module **body parser**

```
npm install body-parser
```

Nous devons ajouter body-parser dans le middleware d'express, dans votre fichier server.js ajoutez :

```
5. const bodyParser = require('body-parser');
6. app.use(bodyParser.urlencoded({extended: false}));
```

Créons maintenant la méthode **process** dans notre Controller **src/controllers/UserRegister.js**

```
1. const UserModel = require('../models/User.js');
2. module.exports = class UserRegister {
3.     print(request, response) {
4.         response.render('user/form_register');
5.     }
6.     process(request, response) {
7.         //console.log(request.body)
8.         let User = new UserModel();
9.         User.add(
10.             request.body.lastname,
11.             request.body.firstname,
12.             request.body.email,
13.             request.body.password
14.         );
```

```
15.     }
16. }
```

Nous devons gérer une réponse pour éviter que le navigateur ne soit en attente, dans le cas de la soumission d'un formulaire, il est courant de faire un **POST-REDIRECT-GET**, c'est à dire que l'on va demander au navigateur de se rediriger vers une autre page. Pour se faire avec express il suffit d'utiliser la méthode redirect, dans le controller.

Après l'insertion de l'utilisateur, ajoutons le redirect dans notre Controller **src/controllers/UserRegister.js** dans la méthode process.

```
17. response.redirect('/');
```

Vérifions tout de même que l'utilisateur (l'email) n'existe pas déjà dans notre base de données !
Nous devons dans un premier temps ajouter la méthode pour vérifier si l'email existe, je vous propose 2 solutions pour le faire, l'utilisation d'un promesse ou l'utilisation d'une méthode asynchrone.

Dans votre classe **src/models/User.js** ajoutez **une** de ces méthodes :

Solution 1) Promise

```
9.      emailExists(email) {
10.         return new Promise((resolve, reject) => {
11.             UserMongo.findOne({ email }, (err, User) => {
12.                 // si pas d'erreur, email trouvé
13.                 if (!err && User !== null) {
14.                    resolve(true);
15.                 }
16.                 resolve(false);
17.             });
18.         });
19.     }
```

Solution 2) Méthode asynchrone

```
9.      async emailExists(email) {
10.         return await UserMongo.findOne({email}) ? true : false;
11.     }
```

Nous devons maintenant dans notre Controller vérifier que l'email n'existe pas avant d'enregistrer l'utilisateur, nous pouvons créer quelques chose comme ça **src/controllers/UserRegister.js** :

```
8.     process(request, response) {
9.         // console.log(request.body)
10.        let User = new UserModel();
11.
12.        User.emailExists(request.body.email).then((result) => {
13.            if(result === false) {
14.                User.add(request.body.lastname,
    request.body.firstname, request.body.email, request.body.password)
15.                // @todo message dans flashbag
16.
17.                // redirection vers l'accueil
18.                response.redirect('/');
19.            }
20.            else {
21.                // @todo reaffichage du formulaire avec message
    d'erreur et donnée dans formulaire
22.                response.render('user/form_register', {
23.                    error : `Cette adresse email est déjà utilisée !`,
24.                    lastname : request.body.lastname,
25.                    firstname : request.body.firstname,
26.                    email : request.body.email
27.                })
28.            }
29.        })
30.    }
```

Nous pouvons voir ligne 18 qu'il y a une redirection vers la page d'accueil dans le cas où l'enregistrement se serait bien passé.

Ligne 22 nous renvoyons le template contenant le formulaire d'inscription en lui transmettant des variables (error, lastname, firstname et email, nous pourrons de ce fait les utiliser dans notre template.

Modifions notre template **templates/user/form_register.pug** pour prendre en compte ces variables :

```
1. extends ../layout.pug
2.
3. block content
4.     h1 Inscription
5.
6.     if error
7.         .alert.alert-danger(role="alert")  #{error}
8.     form(method="post")
9.         .form-group
10.            label(for="firstname") Prénom
11.            input.form-control(type="text" id="firstname"
   name="firstname" placeholder="Votre prénom" required="true"
   value=firstname)
12.        .form-group
13.            label(for="lastname") Nom
14.            input.form-control(type="text" id="lastname"
   name="lastname" placeholder="Votre nom" required="true"
   value=lastname)
15.        .form-group
16.            label(for="email") Adresse E-mail
17.            input.form-control(type="email" id="email" name="email"
   placeholder="Votre adresse email" required="true" value=email)
18.        .form-group
19.            label(for="password")
20.            input.form-control(type="password" id="password"
   name="password" placeholder="Choisissez un mot de passe"
   required="true" )
21.
22.        .form-group
23.            button.btn.btn-primary(type="submit") S'inscrire
```

Notre formulaire fonctionne maintenant parfaitement, il nous reste 2 choses importantes à faire, **hasher le password** car actuellement il est écrit en clair dans la BDD et prévenir l'utilisateur que l'enregistrement c'est bien passé quand il est redirigé vers la page d'accueil, pour cela nous utiliserons les sessions et plus précisément les **flashbag**.

Le hashage du mot de passe

Pour hasher de façon efficace le mot de passe, nous utiliserons le module bcryptjs : https://www.npmjs.com/package/bcryptjs

```
npm install bcryptjs
```

Voici la documentation que nous allons suivre pour le mettre en place : https://www.npmjs.com/package/bcryptjs#usage---async

Modifiez votre méthode add, dans la class User, **src/models/User.js**

```
5.      add(lastname, firstname, email, password) {
6.          let bcrypt = require('bcryptjs');
7.          bcrypt.genSalt(10, function(err, salt) {
8.              bcrypt.hash(password, salt, function(err, hash) {
9.                  return UserMongo.create({
10.                     lastname,
11.                     firstname,
12.                     email,
13.                     password: hash
14.                 })
15.             });
16.         });
17.     }
18.
```

Avec ce code le mot de passe sera hashé de façon efficace.

Ajout de Flashbag

Flashbag est le système utilisé pour écrire un message en session et l'écraser dès la lecture (l'objectif étant de s'assurer que le message ne sera lu qu'une seule fois), ceci est très utilisé pour les messages de confirmation d'un enregistrement (par exemple : "votre compte a bien été créé").

Installez express-flash-messages

```
npm install express-flash-messages
```

Installez express-session

```
npm install express-session
```

Ajoutez dans votre fichier **server.js**

```
9.  //---------------------------------------------------------------------
10. //      Ajout du midlleware express session
11. //---------------------------------------------------------------------
12. const session = require('express-session');
13. app.use(session({
14.     secret: 'key bidon', resave:false, saveUninitialized:false,
15.     cookie: {maxAge: 3600000}
16. }))
17. //---------------------------------------------------------------------
18. //      Ajout du midlleware express flash messages
19. //---------------------------------------------------------------------
20. const flash = require('express-flash-messages')
21. app.use(flash());
```

voir documentation :
https://www.npmjs.com/package/express-flash-messages#usage

Ajoutez dans le fichier **src/controllers/UserRegister.js** dans la méthode process juste avant la redirection :

```
16. request.flash('notify', 'Votre compte a bien été créé.');
```

Dans le fichier **templates/layout.pug** ajoutez les messages

```
17.             - var messages = getMessages()
18.             if messages.notify
19.                 each msg in messages.notify
20.                     .alert.alert-info= msg
21.              if messages.error
22.                 each msg in messages.error
23.                     .alert.alert-danger= msg
```

Connexion utilisateur

Ajoutons la possibilité à nos utilisateurs de se connecter !
Dans le fichier layout.pug

Nous allons ajouter le lien dans notre template de base (**templates/layout.pug**)

```
10.         header
11.             nav
12.                 a(href="/") Accueil
13.                 a(href="/inscription") Inscription
14.                 a(href="/connexion") Connexion
```

Actuellement la route /connexion (url dans notre application) n'existe pas nous devons la créer !

Dans le fichier **app/routes.js** on ajoute la route pour l'inscription (dans la fonction fléchée bien entendu)

```
20.     app.get('/connexion', (req, res) => {
21.         let Authentification =
    require('../src/controllers/Authentification.js')
22.         let Controller = new Authentification();
23.         Controller.print(req, res)
24.     })
```

Cette route fait appel au controller Authentification qui n'existe pas, nous allons donc le créer.

Créez le fichier **src/controllers/Authentification.js**

```
1. module.exports = class Authentification{
2.     print(request, response) {
3.         response.render('user/form_auth')
4.     }
5. }
```

Ce contrôleur fait appel à un template qui n'existe pas, nous devons donc le créer.

Créez le fichier **templates/user/form_auth.pug**

```
1.  extends ../layout.pug
2.
3.  block content
4.      h1 Connexion
5.
6.      form(method="post")
7.          .form-group
8.              label(for="email") Adresse E-mail
9.              input.form-control(type="email" id="email" name="email"
    placeholder="Votre adresse email")
10.         .form-group
11.             label(for="password")
12.             input.form-control(type="password" id="password"
    name="password" placeholder="Choisissez un mot de passe")
13.
14.         .form-group
15.             button.btn.btn-primary(type="submit") Se connecter
```

Prise en compte de la connexion

Nous devons pour prendre en compte la soumission du formulaire de connexion, créer la route en post, dans le fichier **app/routes.js** on ajoute la route pour l'inscription (dans la fonction fléchée bien entendu)

```
27.     app.post('/connexion', (req, res) => {
28.         let Authentification =
    require('../src/controllers/Authentification.js')
29.         let Controller = new Authentification();
30.         Controller.process(req, res)
31.     })
32.
```

Cette route fait appel à la méthode process (qui n'existe pas encore) de notre controller Authentification

Modifiez le fichier **src/controllers/Authentification.js**

```
6.  const UserModel = require('../models/User.js')
7.
8.  module.exports = class Authentification{
9.      print(request, response) {
10.         response.render('user/form_auth')
11.     }
12.
13.     process(request, response) {
14.         let User = new UserModel();
15.         User.connect(request.body.email,
    request.body.password).then(result => {
16.             console.log(result);
17.         });
18.     }
19. }
```

Nous devons vérifier si l'email et le mot de passe sont corrects, je vous propose 2 solutions pour le faire, l'utilisation d'une promesse ou l'utilisation d'une méthode asynchrone.

Dans votre classe **src/models/User.js** ajoutez **une** de ces méthodes :

Solution 1) Promise

```
30.   connect(email, password) {
31.         return new Promise((resolve, reject) => {
32.             UserMongo.findOne({ email }, (err, user) => {
33.                 // si pas d'erreur, email trouvé
34.                 if (!err && user !== null) {
35.                     let bcrypt = require('bcryptjs');
36.                     if(bcrypt.compareSync(password, user.password)) {
37.                         resolve(user);
38.                     }
39.                 }
40.                 resolve(false);
41.             })
42.         })
43.     }
```

Solution 2) Méthode asynchrone

```
30.     async connect(email, password) {
31.         let user = await UserMongo.findOne({email});
32.         if(user !== null) {
33.             let bcrypt = require('bcryptjs');
34.             if(bcrypt.compareSync(password, user.password)) {
35.                 return user;
36.             }
37.         }
38.         return false;
39.     }
```

Nous avons en réponse false si l'identification a échoué et les infos de l'utilisateur si la connexion a réussie, nous devons gérer ces 2 cas !

Si l'identification a échouée nous ré-affichons le template en lui ajoutant un message d'erreur "l'identification a échouée" par exemple.

```
8.   process(request, response) {
9.         let User = new UserModel();
10.        User.connect(request.body.email,
   request.body.password).then(result => {
11.            // l'identification a échouée
12.            if(result == false) {
13.                response.render('user/form_auth', {
14.                    error : `L'identification a échouée`,
15.                    email : request.body.email
16.                });
17.            } else {
18.                // @todo on enregistre les infos en session
19.            }
20.        });
21.    }
```

Nous allons modifier notre template pour qu'il affiche l'erreur si besoin et réécrive l'email dans le champ.

templates/form_auth.pug

```
extends ../layout.pug

block content
    h1 Connexion
    if error
        .alert.alert-danger(role="alert")  #{error}

    form(method="post")
        .form-group
            label(for="email") Adresse E-mail
            input.form-control(type="email" id="email" name="email" placeholder="Votre adresse email" value=email)
        .form-group
            label(for="password")
            input.form-control(type="password" id="password" name="password" placeholder="Choisissez un mot de passe")

        .form-group
            button.btn.btn-primary(type="submit") Se connecter
```

Si l'identification a réussi nous allons stocker les infos en session (id, nom, prenom, email), il est judicieux de créer un niveau spécifiquement pour l'utilisateur dans les sessions (**request.session.user = {};**)
Modifions le else de notre méthode **process** dans le contrôleur **src/controllers/Authentification.js**

```
    // on enregistre les infos en session
    request.session.user = {
        connected : true,
        id : result._id,
        email : result.email,
        isAdmin : result.isAdmin,
        lastname : result.lastname,
        firstname : result.firstname
    };
    // message dans flashbag
```

```
28.     request.flash('notify', 'Vous êtes maintenant connecté.');
29.     // redirection vers l'accueil
30.     response.redirect('/');
```

Afin d'accéder à l'ensemble de nos sessions dans nos templates il faut ajouter un middleware dans notre fichier server.js.
Profitons en aussi pour changer la clef de nos sessions en utilisant un paramètre que nous créerons dans notre fichier app/config.js

Dans **server.js** remplacez le middleware des sessions par :

```
6.  //--------------------------------------------------------------------
7.  //      Ajout du midlleware express session
8.  //--------------------------------------------------------------------
9.  const session = require('express-session');
10. app.use(session({
11.     secret: config.appKey, resave:false, saveUninitialized:false,
12.     cookie: {maxAge: 3600000}
13. }))
14. // permet de renvoyer les sessions à la vue
15. app.use((req,res,next) => {res.locals.session = req.session;
    next();});
```

Ligne 11 nous utilisons un paramètre **(config.appKey)** que nous devons créer dans notre fichier **app/config.js**
Ligne 15 nous créons un middleware pour transmettre aux templates les sessions, nous pourrons ainsi y accéder via la variable **session**.

Nous devons ajouter l'attribut **appKey** dans le fichier **app/config.js**

```
1.  module.exports = {
2.      appKey : '228d8d4ed543ebc79cce4f226b86f5785f21ef07522e5f64',
3.
4.     mongodb :
    'mongodb+srv://username:password@cluster.mongodb.net/ecommerce',
5.      port : 3000
6.  }
```

Pensez à modifier les valeurs en rouge (username:password@cluster) par vos données réelles.

Dans notre fichier **templates/layout.pug** nous allons prendre en considération les sessions, modifiez le **header** :

```
7.
8.          header
9.              nav
10.                 a(href="/") Accueil
11.                 if session.user
12.                     div Bienvenue #{session.user.firstname}
13.                     a(href="/deconnexion") Se déconnecter
14.                 else
15.                     a(href="/inscription") Inscription
16.                     a(href="/connexion") Connexion
```

Nous devons maintenant gérer le bouton de déconnexion.

Ajoutez la route pour la déconnexion, dans le fichier **app/routes.js**

```
33.     app.get('/deconnexion', (req, res) => {
34.         let Authentification =
    require('../src/controllers/Authentification.js')
35.         let Controller = new Authentification();
36.         Controller.deconnect(req, res)
37.     })
```

Ajoutons la méthode **deconnect** à notre controller dans le fichier **src/controllers/Authentification.js**

```
33.     deconnect(request, response) {
34.         delete request.session.user;
35.         // message dans flashbag
36.         request.flash('notify', 'Vous êtes maintenant déconnecté.');
37.         // redirection vers l'accueil
38.         response.redirect('/');
39.     }
```

nous supprimons la session utilisateur, nous créons un message dans les flashbag et nous redirigeons vers la page d'accueil.

Création de l'interface d'administration (il suffit d'être connecté pour être administrateur).

Ajouter le lien vers l'ajout de film, dans le fichier layout.pug

```
20. if session.user
21.     if session.user.isAdmin
22.         li.nav-item
23.             a.nav-link(href="/admin/movie/add/") Ajouter un film
```

Créez ensuite la route (dans app/routes.js) ajoutez :

```
10.     app.get('/admin/movie/add/', (req, res) => {
11.         let AdminMovie = require('../src/controllers/AdminMovie.js')
12.         let Controller = new AdminMovie();
13.         Controller.print(req, res);
14.     });
```

Il faut créer le controller AdminMovie.js

```
1. module.exports = class AdminMovie  {
2.     print(request, response) {
3.         // vérification des droits d'admin
4.         if(this.authAdmin(request, response)) {
5.             response.render('admin/movies');
6.         }
7.     }
8.
9.     authAdmin(request, response) {
10.         if(typeof request.session.user == 'undefined' ||
    request.session.user.isAdmin !== true) {
11.             response.status(401);
12.             response.end('HTTP 401 Unauthorized');
13.             return false;
14.         }
15.         return true;
16.     }
17. }
```

Pour le moment le controller permet uniquement de charger la vue pour printer le formulaire de création de film, vous devez donc créer ce formulaire, voici un exemple : **templates/admin/movies.pug**

```
extends ../layout.pug

block content
    h1 Film

    if error
        .alert.alert-danger(role="alert")  #{error}

    form(method="post")
        .form-group
            label(for="title") Titre
            input.form-control(type="text" id="title" name="title" value=title placeholder="Titre du film")
        .form-group
            label(for="year") Année de production
            input.form-control(type="text" id="year" name="year" value=year pattern="/^(19|20)[0-9]{2}+$/" placeholder="Année de production")
        .form-group
            label(for="genre") Genre
            input.form-control(type="text" id="genre" name="genre" value=genre placeholder="Genre (action, science-fiction, comédie, ...)")
        .form-group
            label(for="actors") Acteur(s)
            input.form-control(type="text" id="actors" name="actors" value=actors placeholder="Liste des acteurs")
        .form-group
            label(for="synopsis")
            textarea.form-control(type="synopsis" id="synopsis" name="synopsis" placeholder="Synopsis")

        .form-group
            button.btn.btn-primary(type="submit") Ajouter
```

Vous devez maintenant gérer l’enregistrement de ce formulaire :
Créer la route en post
Créer le model Movie, créer le model MovieMongoDB qui sera le Model MongoDB pour gérer la collection movies
(appuyez vous sur notre formulaire user pour réussir à le faire par vous même).
Dans le controller Movie ajoutez la méthode qui va gérer l’enregistrement (en faisant appel bien entendu au model).

Dans un premier temps, créons un formulaire pour ajouter des films.
Nous souhaitons avoir au minimum les données suivantes :
Titre, synopsis, année de production (ou réalisation), liste des acteurs (4, ou 5 maximum), affiche du film, genre.
Vous pouvez récupérer d'autres informations si vous le souhaitez.

Nous allons maintenant utiliser l'API pour récupérer les informations des films et les intégrer directement dans le formulaire.
l'API : https://developers.themoviedb.org/3/collections

Créer un compte : https://www.themoviedb.org/signup
Une fois votre compte créé, faites une demande et récupérez votre clé API : https://www.themoviedb.org/settings/api

Ajoutez un attribut **tmdb_api_key** dans **app/config.js** en lui donnant la valeur de votre clef API.

Je vous encourage à enregistrer dans votre BDD mongoDB l'id themoviedb des films que vous enregistrez, afin de pouvoir exécuter d'autres requêtes sur les services par la suite.

Nous allons devoir utiliser plusieurs services de l'API pour un seul film.

En premier nous allons utiliser le service recherche de film :
https://developers.themoviedb.org/3/search/search-movies
Via cette API nous allons récupérer une liste de film en rapport avec la recherche, dans cette liste vous trouverez les ids des films.

Modifiez **admin/movies.pug**

```
1.  extends ../layout.pug
2.
3.  block content
4.      h1 Film
5.      if error
6.          .alert.alert-danger(role="alert")  #{error}
7.
8.      form(method="post")
9.          .form-group
10.             label(for="title") Titre
11.             input.form-control.basicAutoComplete(type="text"
    id="title" name="title" autocomplete="off" value=title
    placeholder="Titre du film")
12.             div#parentMoviesList
13.                 ul#moviesList
14.         .form-group
15.             label(for="year") Année de production
16.             input.form-control(type="number" id="year" name="year"
    min="1900" max=new Date().getFullYear()+1 value=year
    placeholder="Année de production")
17.         .form-group
18.             label(for="genre") Genre
19.             input.form-control(type="text" id="genre" name="genre"
    value=genre placeholder="Genre (action, science-fiction, comédie,
    ...)")
20.         .form-group
21.             label(for="actors") Acteur(s)
22.             input.form-control(type="text" id="actors" name="actors"
    value=actors placeholder="Liste des acteurs")
23.         .form-group
24.             label(for="synopsis")
25.             textarea.form-control(type="synopsis" id="synopsis"
    name="synopsis" placeholder="Synopsis")
26.
27.         .form-group
28.             button.btn.btn-primary(type="submit") Ajouter
29.
30.     template#movieChoice
31.         li
```

```
            img(src="")
            span.title

block javascript
    script(src="/javascript/movie.js")
```

Créez le fichier javascript **public/javascript/movie.js**

```
var typing = false;
var timerTyping = null;
document.addEventListener('DOMContentLoaded',() => {

    document.addEventListener('click', () => {
        document.querySelector('#moviesList').style.display = 'none';
    });

    document.querySelector('.basicAutoComplete').addEventListener('click', (e) => {
        e.stopPropagation();
        if(document.querySelector('#moviesList').childNodes.length > 0) {
            document.querySelector('#moviesList').style.display = 'block';
            document.querySelectorAll('#moviesList li').forEach((li) => {
                li.addEventListener('click', (e) => {
                   console.log(e.currentTarget.dataset.idMovie);
                 });
             });
        }
    });

    document.querySelector('.basicAutoComplete').addEventListener('keypress', (e) => {
        if(typing == false) {
            typing = true;
        } else {
            // sinon on supprime le précédent timer
            window.clearTimeout(timerTyping);
        }
        // dans tous les cas on lance un timer pour executer la recherche
        timerTyping = window.setTimeout(() => {
            typing = false;
            searchMovie(document.querySelector('.basicAutoComplete').value);
```

```
        }, 1000);
    });
});

function searchMovie(query) {
    if(query.length > 3) {
        fetch(`https://api.themoviedb.org/3/search/movie?api_key=dfb4452a8f5050cf53c4cada49f5e037&language=fr-FR&page=1&include_adult=false&query=${query}`)
            .then((response) => response.json())
            .then((data) => {
                document.querySelector('#moviesList').innerHTML = '';
                data.results.forEach((result) => {
                    let template = document.querySelector("#movieChoice");
                    let clone = document.importNode(template.content, true);
                    clone.querySelector('li span.title').textContent = result.title;
                    clone.querySelector('li').dataset.idMovie = result.id;
                    let image = `https://image.tmdb.org/t/p/w200${result.backdrop_path}`;
                    if(result.backdrop_path == null) {
                        image = '/images/no-photo.jpg';
                    }
                    clone.querySelector('li img').src = image;
                    document.querySelector('#moviesList').appendChild(clone);
                });
                document.querySelector('#moviesList').style.display = 'block';
            });
    }
}
```

Quand l'utilisateur choisi un film (en cliquant sur son titre), vous effectuez une nouvelle requete avec l'API sur le service :
https://developers.themoviedb.org/3/movies/get-movie-details
Dans le but de pré-remplir le formulaire avec les informations récupérées.

Vous devez également effectuer une requête sur le service :
https://developers.themoviedb.org/3/movies/get-movie-credits
pour récupérer les informations des acteurs du film.

Vous récupérer le nom de l'image pour récupérer l'url complète suivez cette documentation :
https://developers.themoviedb.org/3/getting-started/images

BONUS:
ajouter côté administration un champ pour la récupération d’urls de vidéo des bandes annonces.
ajouter coté site un player vidéo avec les bandes annonces.