JS codes

Page web dynamique

Index.js

```
/* eslint-disable no-use-before-define */
import 'bootstrap/dist/css/bootstrap.min.css';
import './stylesheets/main.css';

const questions = [];

const questionsAsked = [];

const renderTrainingPage = () => {
    const main = document.querySelector('main');
    main.innerHTML = ``;

    const form = document.createElement('form');

    const labelQuestion = document.createElement('label');
    labelQuestion.textContent = 'Question';

    const inputQuestion = document.createElement('input');
    inputQuestion.type = 'text';
    inputQuestion.id = 'question';
    inputQuestion.setAttribute('required', true);

    const labelReponse = document.createElement('label');
    labelReponse.textContent = 'Reponse';

    const inputReponse = document.createElement('input');
    inputQuestion.type = 'text';
    inputReponse.id = 'reponse'
    inputReponse.setAttribute('required', true);

    const submit = document.createElement('button');
    submit.type = 'submit'
    submit.textContent = 'Enregister exemple'

    form.appendChild(labelQuestion);
    form.appendChild(inputQuestion);

    form.appendChild(labelReponse);
    form.appendChild(inputReponse);
```

```
    form.appendChild(submit);

    main.appendChild(form);

    const button = document.createElement('button');
    button.textContent = 'Etape suivante';
    button.addEventListener('click', () => {
        renderChatBotPage();
    });

    main.appendChild(button);

    form.addEventListener('submit', (event) => {
        event.preventDefault();

        const question = document.querySelector('#question').value;
        const reponse = document.querySelector('#reponse').value;

        const newQuestion = {
            question,
            reponse
        }

        questions.push(newQuestion);

        form.reset();
    });
}

const renderChatBotPage = () => {
    const main = document.querySelector('main');
    main.innerHTML = ``;

    const form = document.createElement('form');

    const labelQuestion = document.createElement('label');
    labelQuestion.textContent = 'Question';

    const inputQuestion = document.createElement('input');
    inputQuestion.type = 'text';
    inputQuestion.id = 'question';
    inputQuestion.setAttribute('required', true);

    const submit = document.createElement('button');
    submit.type = 'submit';
    submit.textContent = 'Poser la question';

    const button = document.createElement('button');
    button.textContent = 'Etape précédente';
```

```
    button.addEventListener('click', () => {
        renderTrainingPage();
    });


    form.appendChild(labelQuestion);
    form.appendChild(inputQuestion);
    form.appendChild(submit);

    main.appendChild(form);

    const div = document.createElement('div');
    main.appendChild(div);

    if (questionsAsked.length !== 0) {
        questionsAsked.forEach((e) => {
            const questionP = document.createElement('p');
            questionP.textContent = `Question: ${e.question}`;

            const reponseP = document.createElement('p');
            reponseP.textContent = `Réponse: ${e.reponse}`

            div.appendChild(questionP);
            div.appendChild(reponseP);
        });
    }

    main.appendChild(button);



    form.addEventListener('submit', (event) => {
        event.preventDefault();

        const question = document.querySelector('#question').value;

        const object = questions.find((e) => e.question === question);

        let reponse = "";
        if (!object) {
            reponse = "Je ne sais pas répondre à cette question";
        } else {
            reponse = object.reponse;
        }

        const questionP = document.createElement('p');
        questionP.textContent = `Question: ${question}`;

        const reponseP = document.createElement('p');
```

```
        reponseP.textContent = `Réponse: ${reponse}`

        div.appendChild(questionP);
        div.appendChild(reponseP)

        form.reset();

        questionsAsked.push({ question, reponse});
    });
}

renderTrainingPage();
```

places.js (utils)

```
/* eslint-disable no-restricted-syntax */
/* eslint-disable no-unused-vars */
/* eslint-disable consistent-return */
/* eslint-disable no-unused-expressions */
import berlin from '../img/berlin.jpg';
import bruges from '../img/bruges.jpg';
import munich from '../img/munich.jpg';
import paris from '../img/paris.jpg';
import rome from '../img/rome.jpg';

const PHOTOS = [
  {
    id: 1,
    name: 'Berlin',
    image: berlin,
  },
  {
    id: 2,
    name: 'Bruges',
    image: bruges,
  },
  {
    id: 3,
    name: 'Munich',
    image: munich,
  },
  {
    id: 4,
    name: 'Paris',
    image: paris,
  },
  {
    id: 5,
```

```
    name: 'Rome',
    image: rome,
  },
];

let place;

const getIdPlace = (id) => {
  let thePlace;
  for (const element of PHOTOS) {
    if (element.id === id)
      thePlace = element;

  }
  return thePlace;

};

const getPlaceWithName = (name) => {
  let result;
  for (const thePlace of PHOTOS) {
    if (thePlace.name === name)
      result = thePlace;

  }
  return result;

};

const getPlace = () => {
  if (place === undefined)
    return;
  return place;

};

const setPlace = (thePlace) => {
  place = thePlace;
};


const isPlace = () => place !== undefined;

const clearPlace = () => {
  place = undefined;
};

export { getPlace, setPlace, isPlace, clearPlace, getPlaceWithName, getIdPlace
};
```

```
// , getNamePlace, getIdPlace
```

RESTful API :

Models :

```
/* eslint-disable no-shadow */
/* eslint-disable import/newline-after-import */
/* eslint-disable no-console */
const path = require('node:path');
const { parse, serialize } = require('../utils/json');
const { users, products } = require("../constants");
const jsonDbPath = path.join(__dirname, '/../data/historique.json');

const defaultHistorique = [];

function readAllUsers() {
  return users;
}

function readAllProducts() {
  return products;
}

function readBetterProduct(id) {
  const idAsNumber = parseInt(id, 10);
  let purchases = parse(jsonDbPath, defaultHistorique);

  purchases = purchases.filter((e) => e.idProduct === idAsNumber);

 // console.log("eeeeeeeeeeeeeeeeeeeeeeeeee", purchases);
  if (purchases.length === 0) return false;

  let max = 0;
  purchases?.forEach((e) => {
      if (e.quantity > max) {
          max = e.quantity;
        // console.log("eeeeeeeeeeeeeeeeeeeeeeeeee", e.quantity);
        // console.log("djddddddddddddddddddddddd", max);
      }
  });
 // console.log("eeeeeeeeeeeeeeeeeeeeeeeeee", max);
  const userFound = purchases.find((e) => e.quantity === max);
```

```
  return userFound.pseudo;
}

function registerOnePurchase(pseudo, idProduct, quantity) {
  const historique = parse(jsonDbPath, defaultHistorique);

  const newHistorique = {
    pseudo,
    idProduct,
    quantity,
  };

  historique.push(newHistorique);
  // console.log("lleleezzzzrfffffffffffff", pseudo, idProduct, quantity);
  serialize(jsonDbPath, historique);

  return newHistorique;
}

function readRecommendationUser(username){
  const allUsers = readAllUsers();
  const userFound = allUsers.find((user) => user === username);
  if (!userFound) return undefined;
  const products = readAllProducts();
  const productRandom = products[Math.floor(Math.random() * products.length)];
  // const productRandom = Math.floor(Math.random() * (products.length - 1)) +
1;
  // return products[productRandom];
  return productRandom;
}


module.exports = {
  readBetterProduct,
  registerOnePurchase,
  readAllProducts,
  readAllUsers,
  readRecommendationUser,
};
```

Routes :

Purchases :

```
/* eslint-disable no-console */
/* eslint-disable import/newline-after-import */
/* eslint-disable no-unused-vars */
const express = require('express');
const router = express.Router();
```

```
const { registerOnePurchase, readAllUsers, readAllProducts, readBetterProduct
} = require("../models/historique");


// Read the pizza identified by an id in the menu
router.get('/:productId', (req, res) => {

    const idInRequest = parseInt(req?.params?.productId, 10);

    const userPseudo = readBetterProduct(idInRequest);

    return res.json(userPseudo);
});


// Create a pizza to be added to the menu.
router.post('/', (req, res) => {
    const pseudo = req?.body?.pseudo;
    const idProduct = req?.body?.idProduct;
    const quantity = req?.body?.quantity;
    const users = readAllUsers();
    if (!users.find((e) => e === pseudo)) return res.status(404).json({
message: "Utilisateur non trouvé" });

    const products = readAllProducts();
    if (!products.find((e) => e.id === idProduct)) return res.sendStatus(400);

    // console.log("lleleezzzzrfffffffffffff", pseudo, idProduct, quantity);
    if (!pseudo || !idProduct || !quantity) return res.sendStatus(400); //
error code '400 Bad request'


    const newPurchase = registerOnePurchase(pseudo, idProduct, quantity);

    return res.json(newPurchase);
});


module.exports = router;
```

Recommendations:

```
/* eslint-disable no-console */
/* eslint-disable import/newline-after-import */
/* eslint-disable no-unused-vars */
const express = require('express');
const router = express.Router();
```

```
const { readRecommendationUser, readAllProducts, readAllUsers } =
require("../models/historique");


router.get('/', (req, res) => {
    const allProducts = readAllProducts();

    return res.json(allProducts);
});

// Read the pizza identified by an id in the menu
router.get('/:username', (req, res) => {
    const userPseudo = req?.params?.username;
    const allUsers = readAllUsers();
    const userFound = allUsers.find((user) => user === userPseudo);
    if (!userFound) return res.sendStatus(400);
    console.log("qddddddddddddddddddddddddd", userPseudo);
    const randomProduct = readRecommendationUser(userPseudo);

    return res.json(randomProduct);
});

module.exports = router;
```

Single Page Application

Exam Aout 2023 Fetch :

```
/* eslint-disable quotes */
/* eslint-disable eol-last */
import { clearPage } from '../../utils/render';

const HomePage = () => {
  clearPage();
  renderHomePagePlaces();
  renderHomePageRecommondedPlace();
};

async function renderHomePagePlaces() {
  const response = await fetch('https://places-exam-
api.azurewebsites.net/places');

  if (!response.ok) throw new Error(`fetch error : ${response.status} :
${response.statusText}`);

  const places = await response.json();
```

```
  const main = document.querySelector('main');
  const h1 = document.createElement('h1');
  h1.innerText = `Tous les lieux`;
  main.appendChild(h1);
  places.forEach((place) => {
    const div = document.createElement('div');
    div.innerHTML = `<div> ${place.name} </div>`;
    main.appendChild(div);
  });
}

async function renderHomePageRecommondedPlace() {
  const response = await fetch('https://places-exam-
api.azurewebsites.net/recommended');

  if (!response.ok) throw new Error(`fetch error : ${response.status} :
${response.statusText}`);

  const place = await response.json();
  const main = document.querySelector('main');
  const h1 = document.createElement('h1');
  h1.innerText = `Lieu recommendé`;
  main.appendChild(h1);
  const div = document.createElement('div');
  div.innerHTML = `<div> ${place.name} </div>`;
  main.appendChild(div);
}

export default HomePage;
```

Méthode HTTP	URI	Opération	Format
GET	/places	Récupérer tous les lieux de vacances	Renvoie : `[ { id, name } ]`
GET	/recommended	Récupérer le lieu de vacances le plus apprécié	Renvoie : `{ id, name }`

Exam Janvier 2023 :

Method	Path	Action	Format
POST	/traduction	Ajoute la traduction d'un mot existant pour entrainer le modèle d'IA	Body : { fr, en }
GET	/traduction/fr?query=value	Traduit un mot du français vers l'anglais	Returns : { fr, en }
GET	/traduction/en?query=value	Traduit un mot de l'anglais vers le français	Returns : { fr, en }

POST :

```
/* eslint-disable prefer-template */
/* eslint-disable no-console */
import { clearPage } from '../../utils/render';

const TrainingPage = () => {
    clearPage();
    renderTrainingPage();

};

async function renderTrainingPage() {
  const main = document.querySelector("main");
    const div = document.createElement('div');

    div.innerHTML = `<form>
  <div class="mb-3">
    <label for="exampleInputEmail1" class="form-label">Français</label>
    <input type="text" class="form-control" id="idFr" aria-
describedby="emailHelp">
  </div>
  <div class="mb-3">
    <label for="exampleInputEmail1" class="form-label">English</label>

    <input type="text" class="form-control" id="idEn" aria-
describedby="emailHelp">
  </div>
  <br>

`;
// const btnRegister = document.querySelector("#btnRegister");
const btnRegister = document.createElement("button");
btnRegister.type = "submit";
btnRegister.className = "btn btn-primary";
btnRegister.textContent = "Ajouter la traduction";

btnRegister.addEventListener('click', async (event) => {
    event.preventDefault();
    const francais = document.querySelector("#idFr").value;
    const english = document.querySelector("#idEn").value;


    const options = {
        method: 'POST',
        body: JSON.stringify({
            fr: francais,
```

```
            en: english
        }),
        headers: {
            'Content-Type': 'application/json',
        },
    };

    const response = await fetch('/api/trad', options);

    if (!response.ok) throw new Error(`fetch error : ${response.status} :
${response.statusText}`);

    const createdTraduction = await response.json();

    console.log("ddddddddddddd"+createdTraduction);
});

    main.appendChild(div);
    main.appendChild(btnRegister);

}

export default TrainingPage;
```

GET :

```
import { clearPage } from '../../utils/render';

const TraductionPage = () => {
  clearPage();
  renderTraductionPage();
};

async function renderTraductionPage() {
    const main = document.querySelector("main");
    const div = document.createElement('div');
    const div2 = document.createElement('div');
    const divTradEn = document.createElement("div");
    const divTradFr = document.createElement("div");

    div.innerHTML = `<form>
  <div class="mb-3">
    <label for="exampleInputEmail1" class="form-label">Français</label>
    <input type="text" class="form-control" id="idFr" aria-
describedby="emailHelp">
  </div>
  <br>
`;
```

```
// const btnRegister = document.querySelector("#btnRegister");
const btnTraduire = document.createElement("button");
btnTraduire.type = "submit";
btnTraduire.className = "btn btn-primary";
btnTraduire.textContent = "Traduire";

div2.innerHTML = ` <div class="mb-3">
<label for="exampleInputEmail1" class="form-label">English</label>
<input type="text" class="form-control" id="idEn" aria-
describedby="emailHelp">
</div>
<br>`;

const btnTranslate = document.createElement("button");
btnTranslate.type = "submit";
btnTranslate.className = "btn btn-primary";
btnTranslate.textContent = "Translate";

const p = document.createElement("p");

btnTraduire.addEventListener('click', async (event) => {
    event.preventDefault();
    const francais = document.querySelector("#idFr").value;

    const requete = await fetch(`/api/trad/fr?query=${francais}`, {
        method: 'GET'
    });

    if (!requete.ok) throw new Error(`fetch error : ${requete.status} :
${requete.statusText}`);
    if (requete.status !== 200) {
        p.innerHTML = `<p>Traduction anglaise: </p><p style="color:
red;">Impossible d'obtenir la traduction</p>`;
    } else {
        const response = await requete.json();
        p.textContent = `Traduction anglaise: ${response.en}`;
    }

    divTradEn.appendChild(p);

});


btnTranslate.addEventListener('click', async (event) => {
    event.preventDefault();
    const english = document.querySelector("#idEn").value;

    const requete = await fetch(`/api/trad/en?query=${english}`, {
        method: 'GET'
```

```
    });

    if (!requete.ok) throw new Error(`fetch error : ${requete.status} :
${requete.statusText}`);
    if (requete.status !== 200) {
        p.innerHTML = `<p>Traduction français: </p><p style="color:
red;">Impossible d'obtenir la traduction</p>`;
    } else {
        const response = await requete.json();
        p.textContent = `Traduction français: ${response.fr}`;
    }

    divTradFr.appendChild(p);

});

    main.appendChild(div);
    main.appendChild(btnTraduire);
    main.appendChild(divTradEn);
    main.appendChild(div2);
    main.appendChild(divTradFr);
    main.appendChild(btnTranslate);
}

export default TraductionPage;
```

TIPS :

Streams :

Find

```
function readBetterProduct(id) {
  const idAsNumber = parseInt(id, 10);
  let purchases = parse(jsonDbPath, defaultHistorique);

  purchases = purchases.filter((e) => e.idProduct === idAsNumber);

 // console.log("eeeeeeeeeeeeeeeeeeeeeeeee", purchases);
  if (purchases.length === 0) return false;

  let max = 0;
  purchases?.forEach((e) => {
      if (e.quantity > max) {
          max = e.quantity;
         // console.log("eeeeeeeeeeeeeeeeeeeeeeeee", e.quantity);
         // console.log("djddddddddddddddddddddddd", max);
      }
```

```
  });
 // console.log("eeeeeeeeeeeeeeeeeeeeeeeeee", max); find
  const userFound = purchases.find((e) => e.quantity === max);

  return userFound.pseudo;
}
```

```
const userFound = allUsers.find((user) => user === username);
```

Filter

```
purchases = purchases.filter((e) => e.idProduct === idAsNumber);
```

Sort and Reverse

```
router.get('/', (req, res) => {
  const orderByTitle = req?.query?.order?.includes('title')
    ? req.query.order
    : undefined;
  let orderedMenu;
  const pizzas = parse(jsonDbPath, MENU);
  if (orderByTitle) orderedMenu = [...pizzas].sort((a, b) =>
a.title.localeCompare(b.title));
  if (orderByTitle === '-title') orderedMenu = orderedMenu.reverse();

  return res.json(orderedMenu ?? pizzas);
});
```

Reduce

const array1 = [1, 2, 3, 4];

// 0 + 1 + 2 + 3 + 4

const initialValue = 0;

const sumWithInitial = array1.reduce((accumulator, currentValue) => accumulator + currentValue, initialValue);

console.log(sumWithInitial);

// Expected output: 10

REST HTTP

Pizza

```
######### NORMAL OPERATION  ###########

### Read all pizzas
GET http://localhost:3000/pizzas

### Read all pizzas with File variable
@baseUrl = http://localhost:3000
GET {{baseUrl}}/pizzas

### Read all pizzas sorted by title (ascending)
GET {{baseUrl}}/pizzas/?order=+title

### Read all pizzas sorted by title (descending)
GET {{baseUrl}}/pizzas/?order=-title

### Read pizza identified by 2
GET {{baseUrl}}/pizzas/2

### Create a pizza
POST {{baseUrl}}/pizzas
Content-Type: application/json

{
    "title":"Magic Green",
    "content":"Epinards, Brocolis, Olives vertes, Basilic"
}

### Delete pizza identified by 2
DELETE {{baseUrl}}/pizzas/2

### Update the pizza identified by 6
PATCH {{baseUrl}}/pizzas/6
Content-Type: application/json

{
    "title":"Magic Green 2"
}




######### ERROR OPERATION  ###########

### Read pizza which does not exists
GET {{baseUrl}}/pizzas/100

### Create a pizza which lacks a property
```

```
POST {{baseUrl}}/pizzas
Content-Type: application/json

{
    "content":"Epinards, Brocolis, Olives vertes, Basilic"
}

### Create a pizza without info for a property
POST {{baseUrl}}/pizzas
Content-Type: application/json

{
    "title":"",
    "content":"Epinards, Brocolis, Olives vertes, Basilic"
}

### Update for a pizza which does not exist
PUT {{baseUrl}}/pizzas/200
Content-Type: application/json

{
    "title":"Magic Green 2"
}

### Update for a pizza which does not provide any info for a property
PUT {{baseUrl}}/pizzas/1
Content-Type: application/json

{
    "title":"Magic Green 2",
    "content":""
}
```

Films

```
@baseUrl = http://localhost:3000

### Read all films
GET {{baseUrl}}/films

### Try to create a film without a token
POST {{baseUrl}}/films/
Content-Type: application/json

{
    "title":"Star Wars: The Phantom Menace (Episode I)",
    "duration": 136,
    "budget": 115,
```

```
    "link":"https://en.wikipedia.org/wiki/Star_Wars:_Episode_I_%E2%80%93_The_Phantom_Menace"
}

### Create a film with guest token
#### Login the guest user and get the response in a request variable named 'guest'
# @name guest
POST {{baseUrl}}/auths/login
Content-Type: application/json

{
    "username":"guest",
    "password":"guest"
}

#### Create a pizza
POST {{baseUrl}}/films/
Content-Type: application/json
Authorization: {{guest.response.body.token}}

{
    "title":"Star Wars: The Phantom Menace (Episode I)",
    "duration": 136,
    "budget": 115,
    "link":"https://en.wikipedia.org/wiki/Star_Wars:_Episode_I_%E2%80%93_The_Phantom_Menace"
}

### Try to create a film with a parameter missing or empty string or string with whitespaces only
POST {{baseUrl}}/films/
Content-Type: application/json
Authorization: {{guest.response.body.token}}

{
    "title":" ",
    "duration": 136,
    "budget": 115,
    "link":"https://en.wikipedia.org/wiki/Star_Wars:_Episode_I_%E2%80%93_The_Phantom_Menace"
}

### Try to create a film with a wrong budget
POST {{baseUrl}}/films/
Content-Type: application/json
Authorization: {{guest.response.body.token}}

{
```

```
    "title":"Star Wars: The Phantom Menace (Episode I)",
    "duration": 136,
    "budget": "115",
    "link":"https://en.wikipedia.org/wiki/Star_Wars:_Episode_I_%E2%80%93_The_Phantom_Menace"
}

### Create another film
POST {{baseUrl}}/films/
Content-Type: application/json
Authorization: {{guest.response.body.token}}

{
    "title":"Star Wars: Episode 2",
    "duration": 1,
    "budget": 11,
    "link":"findIt.com"
}

### Read film with ID == 1
GET {{baseUrl}}/films/1


### Update film with ID == 2
PATCH {{baseUrl}}/films/2
Content-Type: application/json
Authorization: {{guest.response.body.token}}

{
    "title":"Star Wars: Episode II – Attack of the Clones",
    "duration": 142,
    "budget": 115,
    "link":"https://en.wikipedia.org/wiki/Star_Wars:_Episode_II_%E2%80%93_Attack_of_the_Clones"
}

### Delete the film with ID == 2
DELETE {{baseUrl}}/films/2
Authorization: {{guest.response.body.token}}


### Read all films with minimum duration of 140 minutes
GET {{baseUrl}}/films?minimum-duration=140

### Create a long film
POST {{baseUrl}}/films/
Content-Type: application/json
Authorization: {{guest.response.body.token}}
```

```
{
    "title":"Zack Snyder's Justice League",
    "duration": 242,
    "budget": 70,
    "link":"https://en.wikipedia.org/wiki/Zack_Snyder%27s_Justice_League"
}
```

Models :

```
const path = require('node:path');
const { parse, serialize } = require('../utils/json');

const jsonDbPath = path.join(__dirname, '/../data/films.json');

function readAllFilms(minimumDuration) {
  const films = parse(jsonDbPath);

  if (minimumDuration === undefined) return films;

  const minimumDurationAsNumber = parseInt(minimumDuration, 10);
  if (Number.isNaN(minimumDurationAsNumber) || minimumDurationAsNumber < 0)
return undefined;

  const filmsReachingMinimumDuration = films.filter((film) => film.duration >=
minimumDuration);
  return filmsReachingMinimumDuration;
}

function readOneFilm(id) {
  const idAsNumber = parseInt(id, 10);
  const films = parse(jsonDbPath);
  const indexOfFilmFound = films.findIndex((pizza) => pizza.id ===
idAsNumber);
  if (indexOfFilmFound < 0) return undefined;

  return films[indexOfFilmFound];
}

function createOneFilm(title, link, duration, budget) {
  const films = parse(jsonDbPath);

  const createdPizza = {
    id: getNextId(),
    title,
    link,
    duration,
    budget,
```

```
  };

  films.push(createdPizza);

  serialize(jsonDbPath, films);

  return createdPizza;
}

function getNextId() {
  const films = parse(jsonDbPath);
  const lastItemIndex = films?.length !== 0 ? films.length - 1 : undefined;
  if (lastItemIndex === undefined) return 1;
  const lastId = films[lastItemIndex]?.id;
  const nextId = lastId + 1;
  return nextId;
}

function deleteOneFilm(id) {
  const idAsNumber = parseInt(id, 10);
  const films = parse(jsonDbPath);
  const foundIndex = films.findIndex((pizza) => pizza.id === idAsNumber);
  if (foundIndex < 0) return undefined;
  const deletedFilms = films.splice(foundIndex, 1);
  const deletedFilm = deletedFilms[0];
  serialize(jsonDbPath, films);

  return deletedFilm;
}

function updateOneFilm(id, propertiesToUpdate) {
  const idAsNumber = parseInt(id, 10);
  const films = parse(jsonDbPath);
  const foundIndex = films.findIndex((pizza) => pizza.id === idAsNumber);
  if (foundIndex < 0) return undefined;

  const updatedPizza = { ...films[foundIndex], ...propertiesToUpdate };

  films[foundIndex] = updatedPizza;

  serialize(jsonDbPath, films);

  return updatedPizza;
}

module.exports = {
  readAllFilms,
  readOneFilm,
  createOneFilm,
```

```
  deleteOneFilm,
  updateOneFilm,
};
```

Routes :

```
const express = require('express');
const {
  readAllFilms,
  readOneFilm,
  createOneFilm,
  deleteOneFilm,
  updateOneFilm,
} = require('../models/films');
const { authorize } = require('../utils/auths');

const router = express.Router();

// Read all the films, filtered by minimum-duration if the query param exists
router.get('/', (req, res) => {
  const filmsPotentiallyFiltered = readAllFilms(req?.query?.['minimum-
duration']);

  if (filmsPotentiallyFiltered === undefined) return res.sendStatus(400);

  return res.json(filmsPotentiallyFiltered);
});

// Read a film from its id in the menu
router.get('/:id', (req, res) => {
  const foundFilm = readOneFilm(req?.params?.id);

  if (!foundFilm) return res.sendStatus(404);

  return res.json(foundFilm);
});

// Create a film
router.post('/', authorize, (req, res) => {
  const title = req?.body?.title?.trim()?.length !== 0 ? req.body.title :
undefined;
  const link = req?.body?.content?.trim().length !== 0 ? req.body.link :
undefined;
  const duration =
    typeof req?.body?.duration !== 'number' || req.body.duration < 0
      ? undefined
      : req.body.duration;
```

```
  const budget =
    typeof req?.body?.budget !== 'number' || req.body.budget < 0 ? undefined :
req.body.budget;

  if (!title || !link || !duration || !budget) return res.sendStatus(400);

  const createdFilm = createOneFilm(title, link, duration, budget);

  return res.json(createdFilm);
});

// Delete a film
router.delete('/:id', authorize, (req, res) => {
  const deletedFilm = deleteOneFilm(req?.params?.id);

  if (!deletedFilm) return res.sendStatus(404);

  return res.json(deletedFilm);
});

// Update a film identified by its id
router.patch('/:id', authorize, (req, res) => {
  const title = req?.body?.title;
  const link = req?.body?.link;
  const duration = req?.body?.duration;
  const budget = req?.body?.budget;

  if (
    !req.body ||
    (title && !title.trim()) ||
    (link && !link.trim()) ||
    (duration && (typeof req?.body?.duration !== 'number' || duration < 0)) ||
    (budget && (typeof req?.body?.budget !== 'number' || budget < 0))
  )
    return res.sendStatus(400);

  const updatedFilm = updateOneFilm(req?.params?.id, req?.body);

  if (!updatedFilm) return res.sendStatus(404);

  return res.json(updatedFilm);
});

module.exports = router;
```

GET **films?minimum-duration=value**

Jokes.js (utils)

```
const jokes = [
  {
    "id": 1,
    "question": "Why are modern programming languages so materialistic?",
    "answer": "Because they are object-oriented.",
    "category": "Programming"
  }
  ,
  {"id": 2,
    "question": "What's the object-oriented way to become wealthy?",
    "answer": "Inheritance.",
    "category": "Programming"
  }
  ,
  {
    "id": 3,
    "question": "What did the fish say when it swam into the wall?",
    "answer": "Dam.",
    "category": "Pun"
  }
  ,
  {
    "id": 4,
    "question": "How much did your chimney cost?",
    "answer": "Nothing, it was on the house.",
    "category": "Pun"
  }
  ,
  {
    "id": 5,
    "question": "Who is Santa's favourite singer?",
    "answer": "Elf-is Presley!",
    "category": "Christmas"
  }
  ,
  {
    "id": 6,
    "question": "What's Santa's favourite type of music?",
    "answer": "Wrap!",
    "category": "Christmas"
  }
]


let joke;

const allJokes = () => {
```

```
  return jokes;

};

const getJoke = () => {
  if (joke === undefined)
    return;
  return joke;

};

function getJokeWithCategory(category)  {
  const theAllJokes = allJokes();
  // theAllJokes.sort((e) => e.category === category);
  let jokeSorted = [];
  for (let i = 0; i < theAllJokes.length; i++) {
    if(theAllJokes[i].category === category){
      jokeSorted[i] = theAllJokes[i];
    }
  }

 // console.log("mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm"+ theAllJokes);
  return jokeSorted;
};

const setJoke = (theJoke) => {
  joke = theJoke;
};


const isJoke = () => joke !== undefined;

const clearJoke = () => {
  joke = undefined;
};

export { allJokes, getJoke, setJoke, isJoke, clearJoke, getJokeWithCategory };
```