جـامعـة الحسن الثـاني بالدار البيـضـاء
UNIVERSITÉ HASSAN II DE CASABLANCA

ECOLE NORMALE SUPERIEURE
DE L'ENSEIGNEMENT TECHNIQUE
ENSET Mohammedia

DÉPARTEMENT MATHÉMATIQUE INFORMATIQUE

Technologie Web

Rapport

PROJET DE FIN DE MODULE

Réaliser par :

ETOULLALI Ayoub

Professeur :

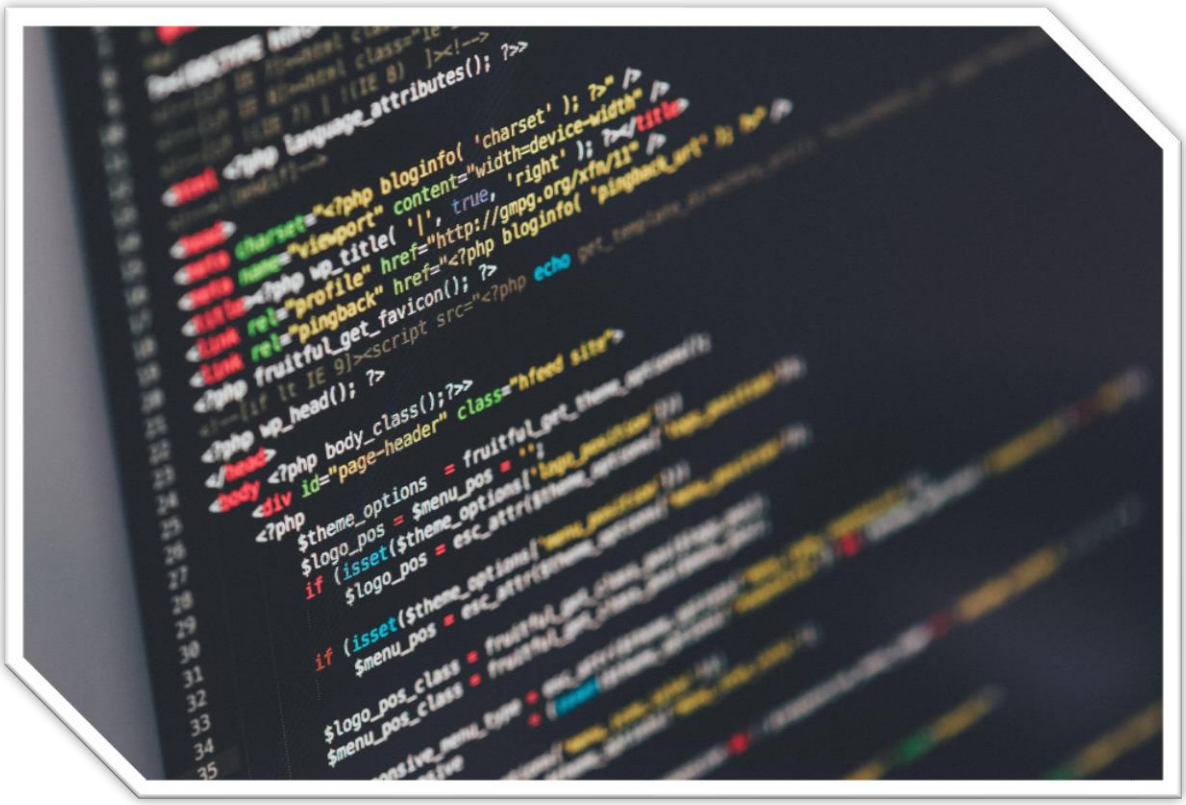
Aziz DAAIF

1ere année II-BDCC

Filière d'ingénieur : Ingénieur informatique, Big Data et Cloud Computing

2021/2022

SOMMAIRE



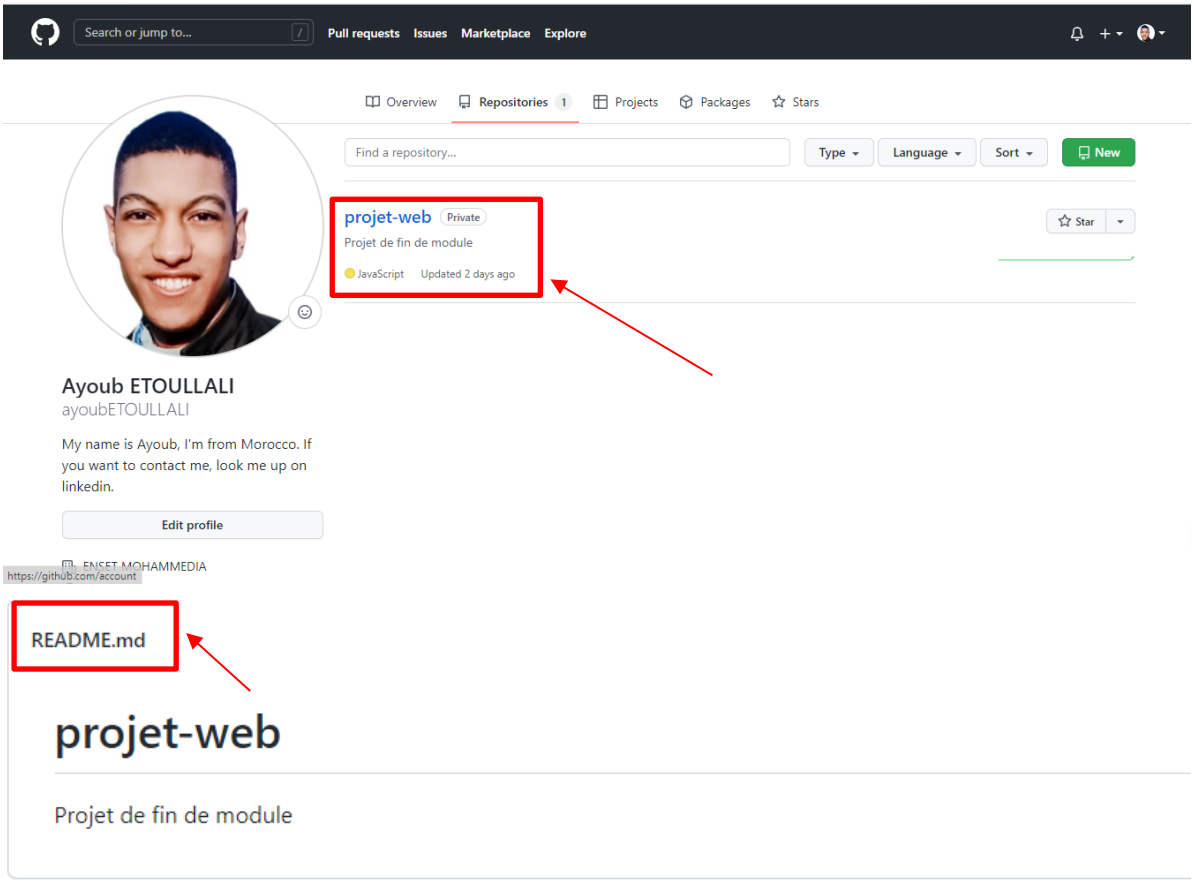
Un projet web est un ensemble d'activités menées par l'équipe projet digitale, sous la supervision du chef de projet, dans le but d'apporter une solution à un besoin précis.
Cela peut être une application web, un site vitrine, un site e-commerce, un blog, un forum ou une application mobile.

Dans ce rapport, je vais vous présenter toutes les étapes de réalisation d'un projet web qui se compose d'un blog sur la photographie.

BACKEND

1. Créer un projet dans VS Code dans un dossier nommé **projet-web**.

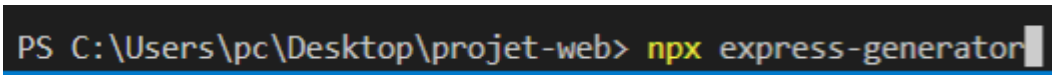
2. Initiez le suivi du projet à l'aide de **git** (git init), ajoutez un fichier **README.md** puis faites un premier commit. Synchronisez ensuite, votre projet avec le dépôt distant correspondant.



3. Initiez dans ce dossier, un projet **Express** en utilisant la commande "**express**"

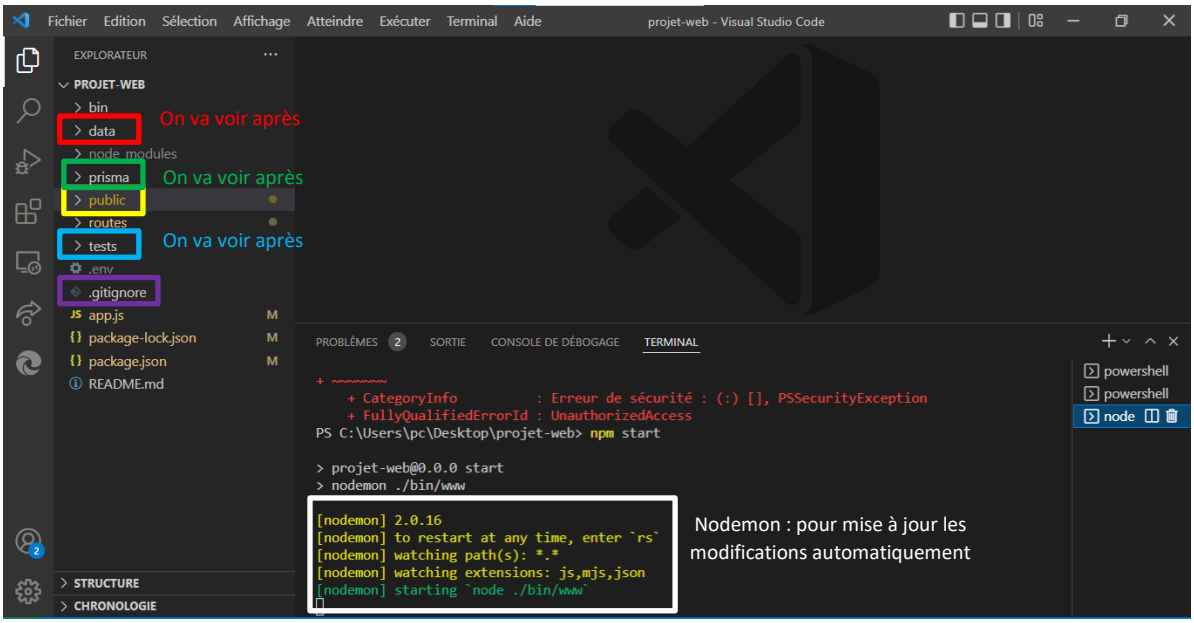
   Puis installez les dépendances.

   Commande :

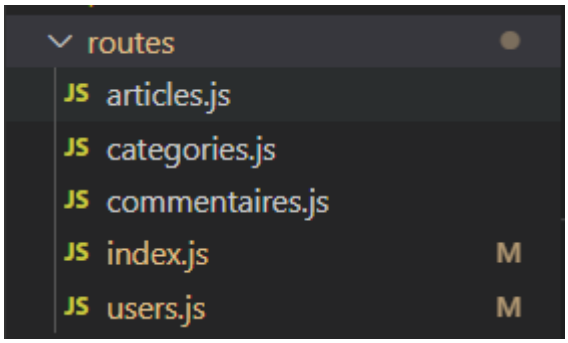


4. Ajoutez dans la racine du projet un fichier nommé "**.gitignore**" contenant "**node_modules**".

   (Pour éviter d'envoyer ce dossier au dépôt distant)



6. Dans le dossier routes, ajouter trois fichiers :
   a. articles.js
   b. categories.js
   c. commentaires.js



7. Pour chacun des fichiers (articles.js, categories.js, commentaires.js et users.js), ajouter les routes permettant d'effectuer les opérations de base CRUD sans prendre en considération les relations entre les entités :

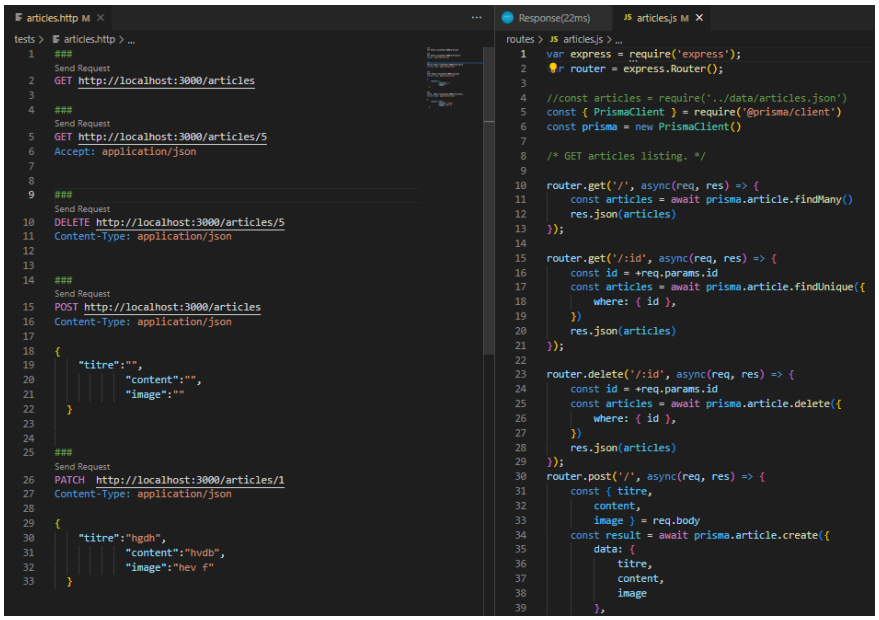
Articles (chemin de base /articles)

Méthode HTTP	Path	Paramètres d'URL	Commentaire
GET	**/**	**take, skip**	Récupérer **take articles** à partir de la position **skip**. take (nombre d'éléments) skip (offset à partir de laquelle on extrait les données de la base)
GET	**/:id**		Récupérer un article ayant l'id donné
POST	**/**		Ajouter un nouveau article envoyé sous format JSON
PATCH	**/**		Mettre à jour l'article envoyé dans le corps de la requête.
DELETE	**/:id**		Supprimer l'article ayant l'id donné.

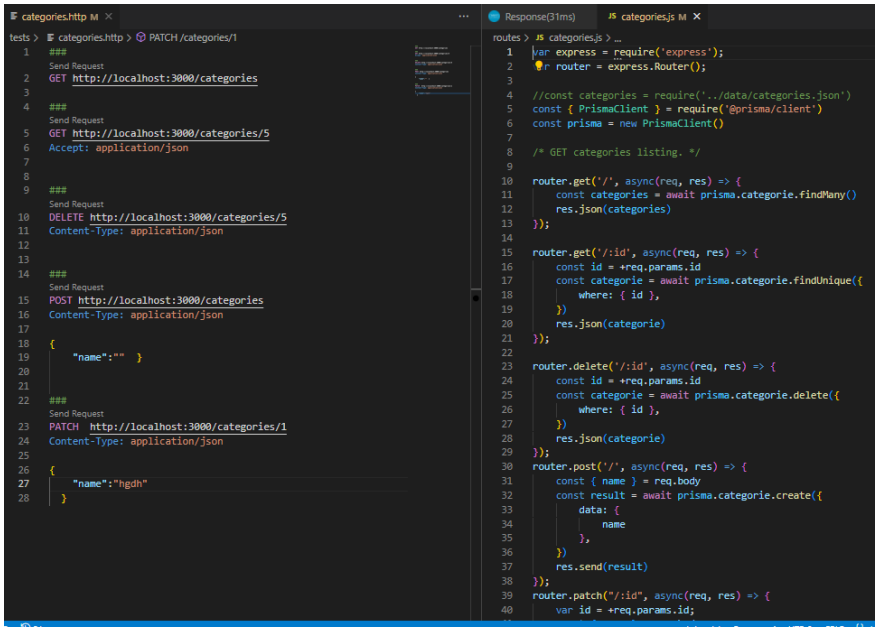
Suivre le même modèle pour les autres routes.
Pour tester les routes, créer un dossier test et mettez dedans les fichiers suivants :

Des exemples pour assurer l'exécution de programme

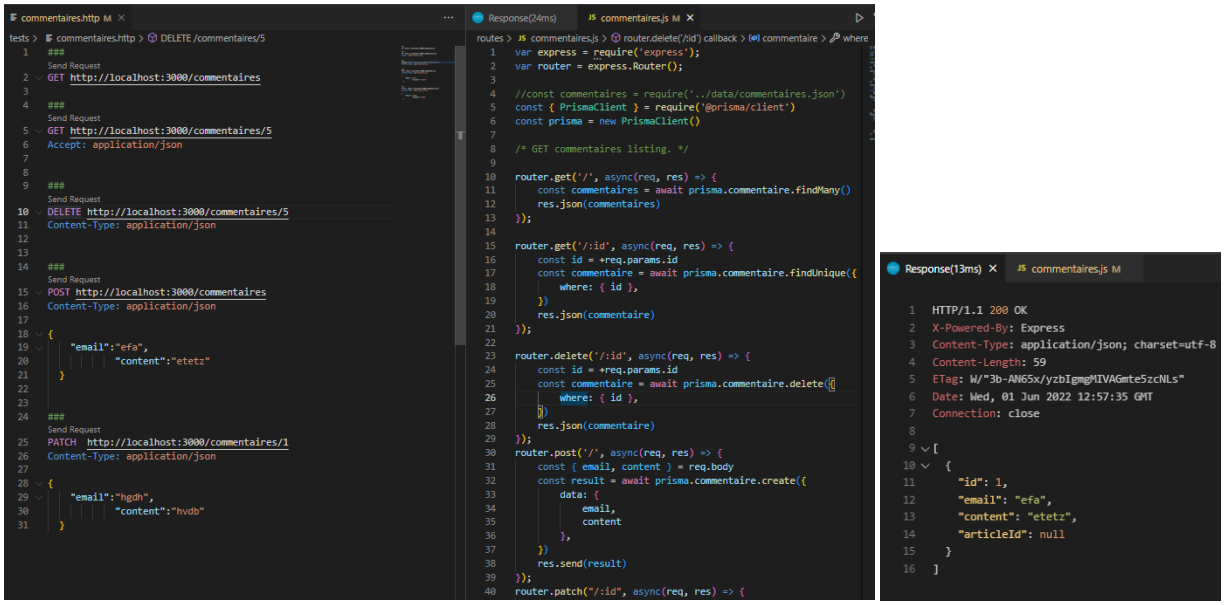
- articles.http



- categories.http



- commentaires.http

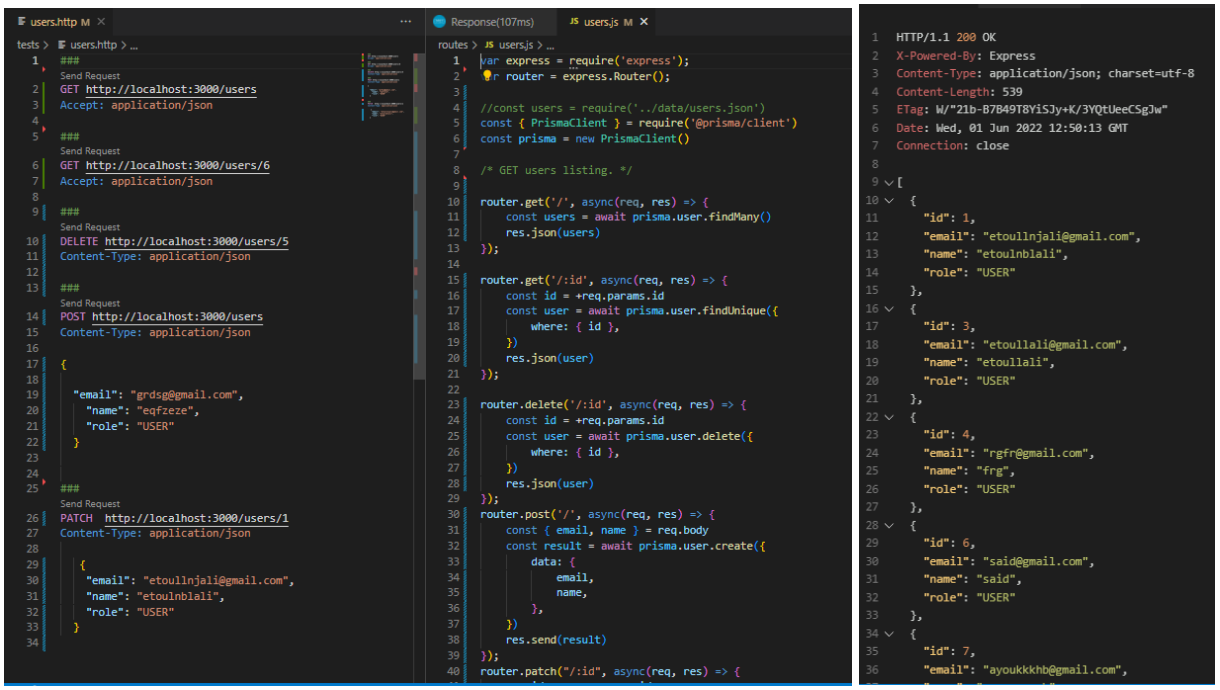


```
### Send Request
GET http://localhost:3000/commentaires

### Send Request
GET http://localhost:3000/commentaires/5
Accept: application/json

### Send Request
DELETE http://localhost:3000/commentaires/5
Content-Type: application/json

### Send Request
POST http://localhost:3000/commentaires
Content-Type: application/json

{
    "email":"efa",
    "content":"etetz"
}

### Send Request
PATCH  http://localhost:3000/commentaires/1
Content-Type: application/json

{
    "email":"hgdh",
    "content":"hvdb"
}
```

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

//const commentaires = require('../data/commentaires.json')
const { PrismaClient } = require('@prisma/client')
const prisma = new PrismaClient()

/* GET commentaires listing. */

router.get('/', async(req, res) => {
    const commentaires = await prisma.commentaire.findMany()
    res.json(commentaires)
});

router.get('/:id', async(req, res) => {
    const id = +req.params.id
    const commentaire = await prisma.commentaire.findUnique({
        where: { id },
    })
    res.json(commentaire)
});

router.delete('/:id', async(req, res) => {
    const id = +req.params.id
    const commentaire = await prisma.commentaire.delete({
        where: { id },
    })
    res.json(commentaire)
});
router.post('/', async(req, res) => {
    const { email, content } = req.body
    const result = await prisma.commentaire.create({
        data: {
            email,
            content
        },
    })
    res.send(result)
});
router.patch("/:id", async(req, res) => {
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 59
ETag: W/"3b-AN65x/yzbIgmgMIVAGmte5zcNLs"
Date: Wed, 01 Jun 2022 12:57:35 GMT
Connection: close

[
  {
    "id": 1,
    "email": "efa",
    "content": "etetz",
    "articleId": null
  }
]
```

- users.http



```
### Send Request
GET http://localhost:3000/users
Accept: application/json

### Send Request
GET http://localhost:3000/users/6
Accept: application/json

### Send Request
DELETE http://localhost:3000/users/5
Content-Type: application/json

### Send Request
POST http://localhost:3000/users
Content-Type: application/json

{
  "email": "grdsg@gmail.com",
  "name": "eqfzeze",
  "role": "USER"
}

### Send Request
PATCH  http://localhost:3000/users/1
Content-Type: application/json

  {
    "email": "etoullnjali@gmail.com",
    "name": "etoulnblali",
    "role": "USER"
  }
```

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

//const users = require('../data/users.json')
const { PrismaClient } = require('@prisma/client')
const prisma = new PrismaClient()

/* GET users listing. */

router.get('/', async(req, res) => {
    const users = await prisma.user.findMany()
    res.json(users)
});

router.get('/:id', async(req, res) => {
    const id = +req.params.id
    const user = await prisma.user.findUnique({
        where: { id },
    })
    res.json(user)
});

router.delete('/:id', async(req, res) => {
    const id = +req.params.id
    const user = await prisma.user.delete({
        where: { id },
    })
    res.json(user)
});
router.post('/', async(req, res) => {
    const { email, name } = req.body
    const result = await prisma.user.create({
        data: {
            email,
            name,
        },
    })
    res.send(result)
});
router.patch("/:id", async(req, res) => {
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 539
ETag: W/"21b-B7B49T8YiSJy+K/3YQtUeeCSgJw"
Date: Wed, 01 Jun 2022 12:50:13 GMT
Connection: close

[
  {
    "id": 1,
    "email": "etoullnjali@gmail.com",
    "name": "etoulnblali",
    "role": "USER"
  },
  {
    "id": 3,
    "email": "etoullali@gmail.com",
    "name": "etoullali",
    "role": "USER"
  },
  {
    "id": 4,
    "email": "rgfr@gmail.com",
    "name": "frg",
    "role": "USER"
  },
  {
    "id": 6,
    "email": "said@gmail.com",
    "name": "said",
    "role": "USER"
  },
  {
    "id": 7,
    "email": "ayoukkkhb@gmail.com",
```

✓ **Les exécutions de GET, POST, PATCH et DELETE marche bien**

9. Installez la **CLI Prisma** et le **Client Prisma**

> **npm i -D prisma**

```
PS C:\Users\pc\Desktop\projet-web> npm i -D prisma
```

```
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'fix@0.0.3',
npm WARN EBADENGINE   required: { node: '0.2.5' },
npm WARN EBADENGINE   current: { node: 'v17.9.0', npm: '8.5.5' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'pipe@0.0.1',
npm WARN EBADENGINE   required: { node: '0.2.5' },
npm WARN EBADENGINE   current: { node: 'v17.9.0', npm: '8.5.5' }
npm WARN EBADENGINE }

up to date, audited 72 packages in 6s

found 0 vulnerabilities
```

> **npm i @prisma/client**

```
PS C:\Users\pc\Desktop\projet-web> npm i @prisma/client
```

```
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'fix@0.0.3',
npm WARN EBADENGINE   required: { node: '0.2.5' },
npm WARN EBADENGINE   current: { node: 'v17.9.0', npm: '8.5.5' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: 'pipe@0.0.1',
npm WARN EBADENGINE   required: { node: '0.2.5' },
npm WARN EBADENGINE   current: { node: 'v17.9.0', npm: '8.5.5' }
npm WARN EBADENGINE }

up to date, audited 72 packages in 6s

found 0 vulnerabilities
```

10. Initiez un dans la racine de votre projet la prise en charge de l'ORM Prisma :

    > **npx prisma init**

```
PS C:\Users\pc\Desktop\projet-web> npx prisma init
```

11. Dans le fichier "**.env**", configurer l'utilisation d'un SGBBR MySQL.

    Assurez-vous d'avoir un serveur MySQL démarré.

```
.env
1   DATABASE_URL="mysql://root:@localhost/projet-web"
```

12. Ajouter dans le fichier "**prisma/schema.prisma**", les modèles reflétant les entités et les relations entre elles.

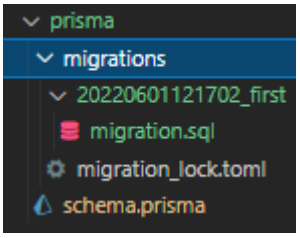
Compléter la configuration pour la prise en charge de **mysql**.

schema.prisma M

prisma > schema.prisma

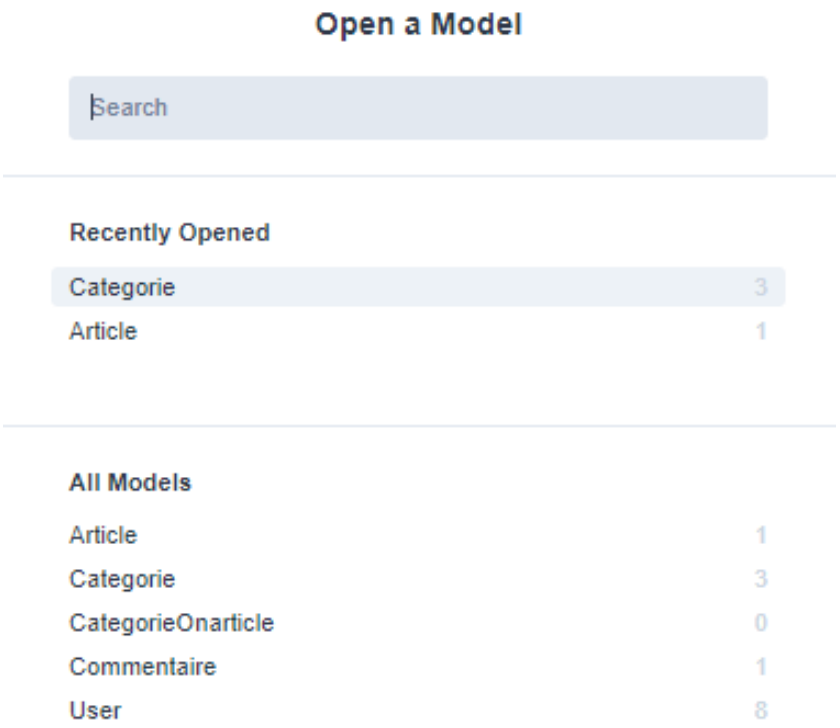
```prisma
// This is your Prisma schema file,
// learn more about it in the docs: https://pris.ly/d/prisma-schema

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "mysql"
  url      = env("DATABASE_URL")
}

model User {
  id        Int      @id @default(autoincrement())
  email     String   @unique @db.VarChar(30)
  name      String?  @db.VarChar(20)
  role      Role     @default(USER)
  article   Article[]
}

model Article {
  id           Int     @id @default(autoincrement())
  titre        String  @db.VarChar(255)
  content      String
  image        String
  createdAt  DateTime @default(now())
  updatedAt  DateTime @updatedAt
  published   Boolean @default(false)
  user User? @relation(fields: [userId], references: [id])
  userId Int?
  categorie CategorieOnarticle[]
  commentaire   Commentaire[]
}
enum Role {
  USER
  ADMIN
  AUTHOR
}
model Categorie {
  id        Int      @id @default(autoincrement())
  name      String? @db.VarChar(20)
  article CategorieOnarticle[]
}

model Commentaire {
  id      Int    @id @default(autoincrement())
  email     String   @db.VarChar(30)
  content  String  @db.VarChar(500)
  article  Article? @relation(fields: [ articleId], references: [id])
  articleId Int?

}
model CategorieOnarticle {
  article       Article     @relation(fields: [articleId], references: [id])
  articleId     Int // relation scalar field (used in the `@relation` attribute above)
  categorie   Categorie  @relation(fields: [categorieId], references: [id])
  categorieId Int // relation scalar field (used in the `@relation` attribute above)
  assignedAt DateTime @default(now())
  assignedBy String

  @@id([articleId, categorieId])
}
```

13. Effectuez les migrations puis régénérer le client Prisma (migrate/generate) et enfin vérifier en ouvrant prisma studio.



> **npx prisma studio**

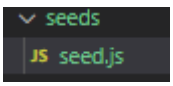
```
PS C:\Users\pc\Desktop\projet-web> npx prisma studio
Environment variables loaded from .env
Prisma schema loaded from prisma\schema.prisma
Prisma Studio is up on http://localhost:5555
```



15. Revenir à express et compléter les routes en y ajoutant le code permettant d'accéder à MySQL via l'ORM Prisma. Puis faire des tests

Voir les images ci-dessus

16. Dans la racine de votre projet, créer le fichier "**seeds/seed.js**".



Avant, il doit installer **faker**

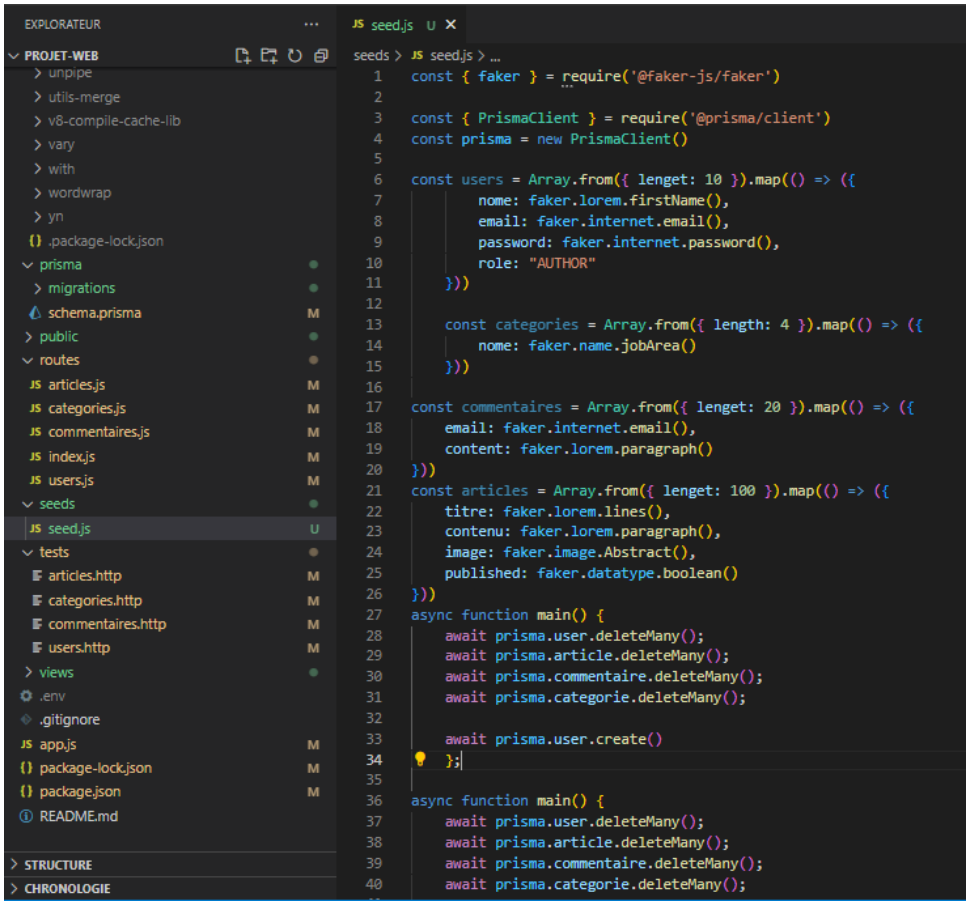
```
PS C:\Users\pc\Desktop\projet-web> npm install @faker-js/faker --force
```

Ce fichier nous permettra lorsqu'il sera exécuté

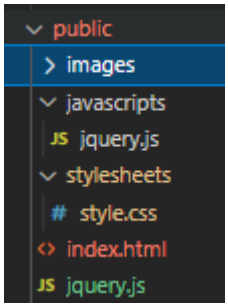
```
PS C:\Users\pc\Desktop\projet-web> node ./seeds/seed.js
```

de créer en utilisant la bibliothèque “Faker” des données de tests. Vous devez créer :

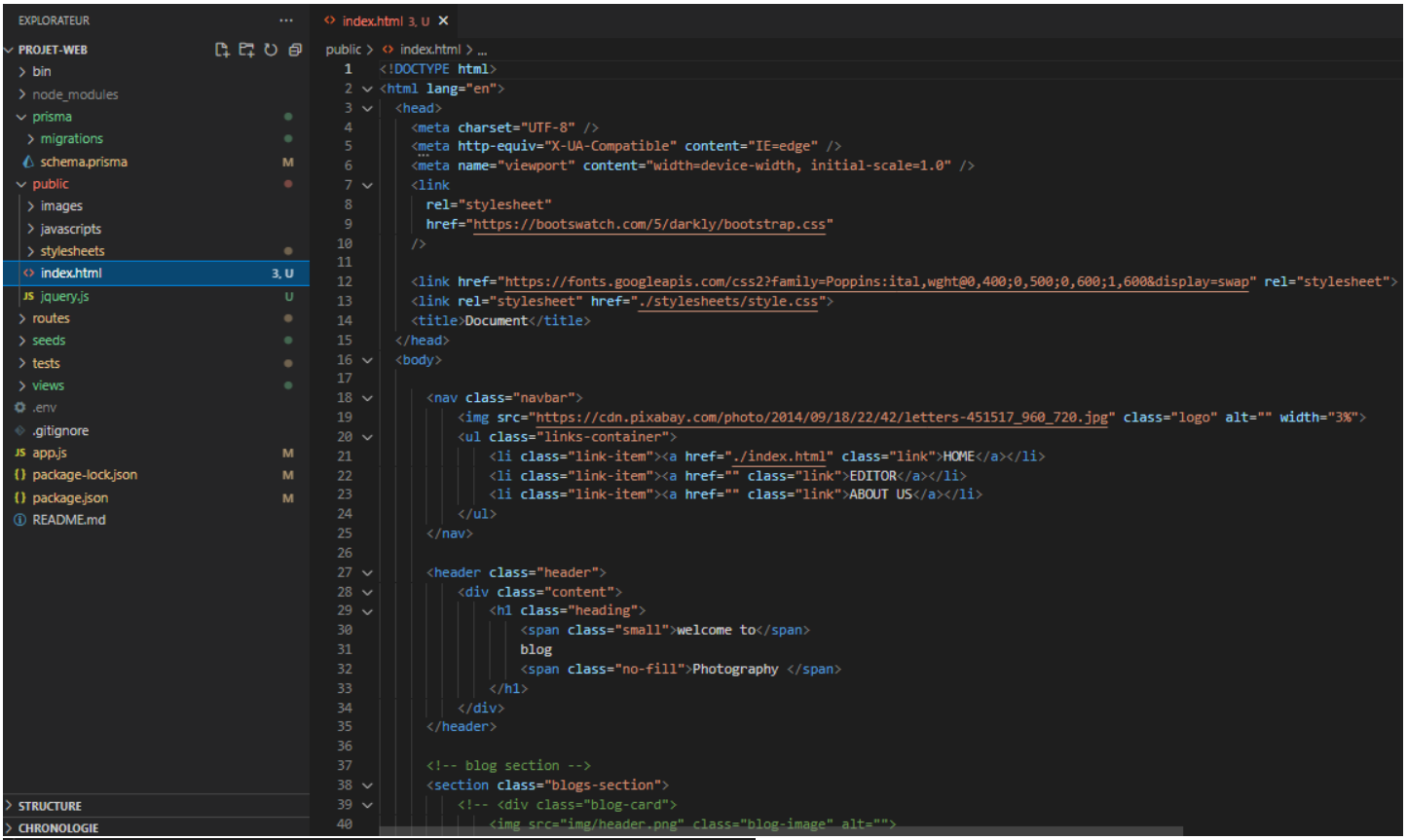
- 10 utilisateurs ayant le rôle “AUTHOR”
- 1 utilisateur ayant le rôle “ADMIN”
- 10 catégories
- 100 articles appartenant à (de 1 à 4 catégories aléatoires) et écrit par l’un des 10 utilisateurs (AUTHOR)
- Pour chaque article, créer de 0 à 20 commentaires



Dossier publlic

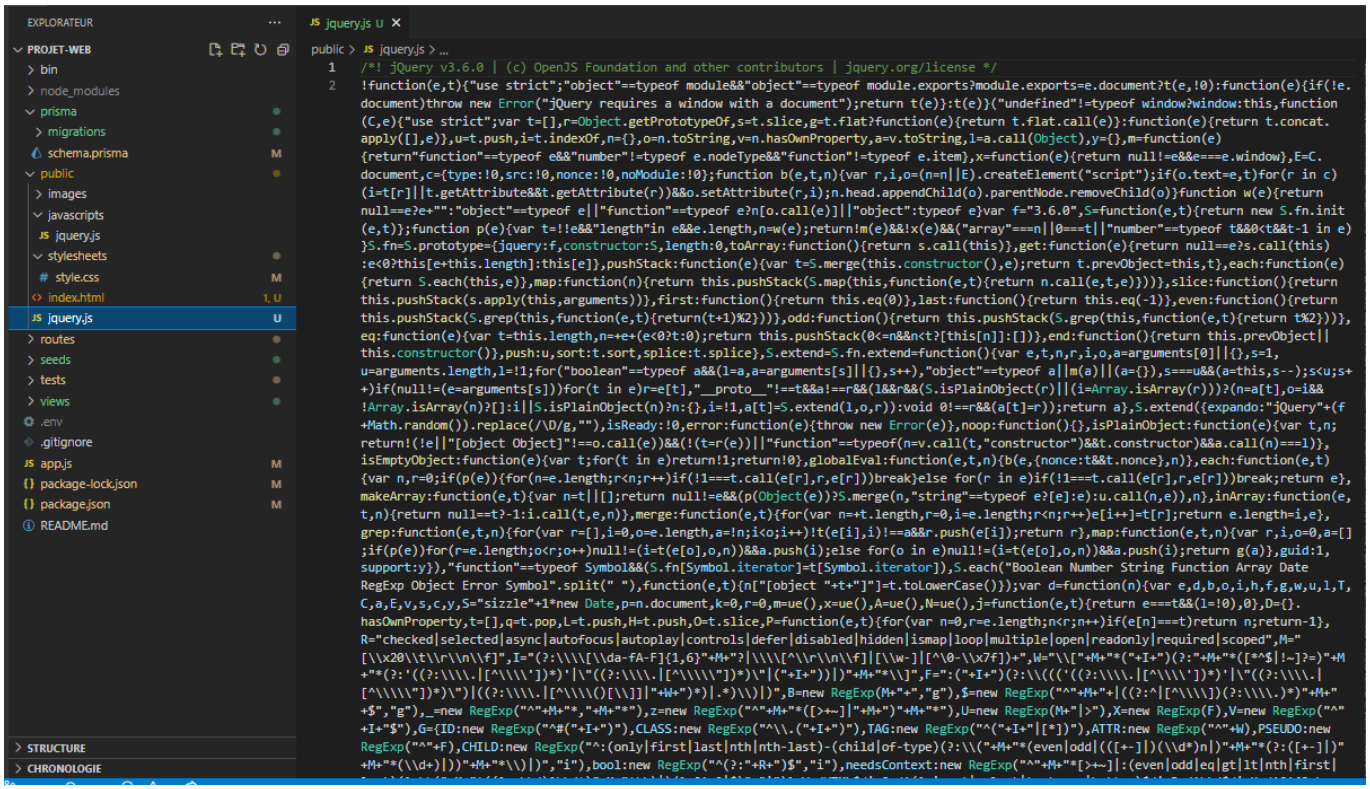


index.html

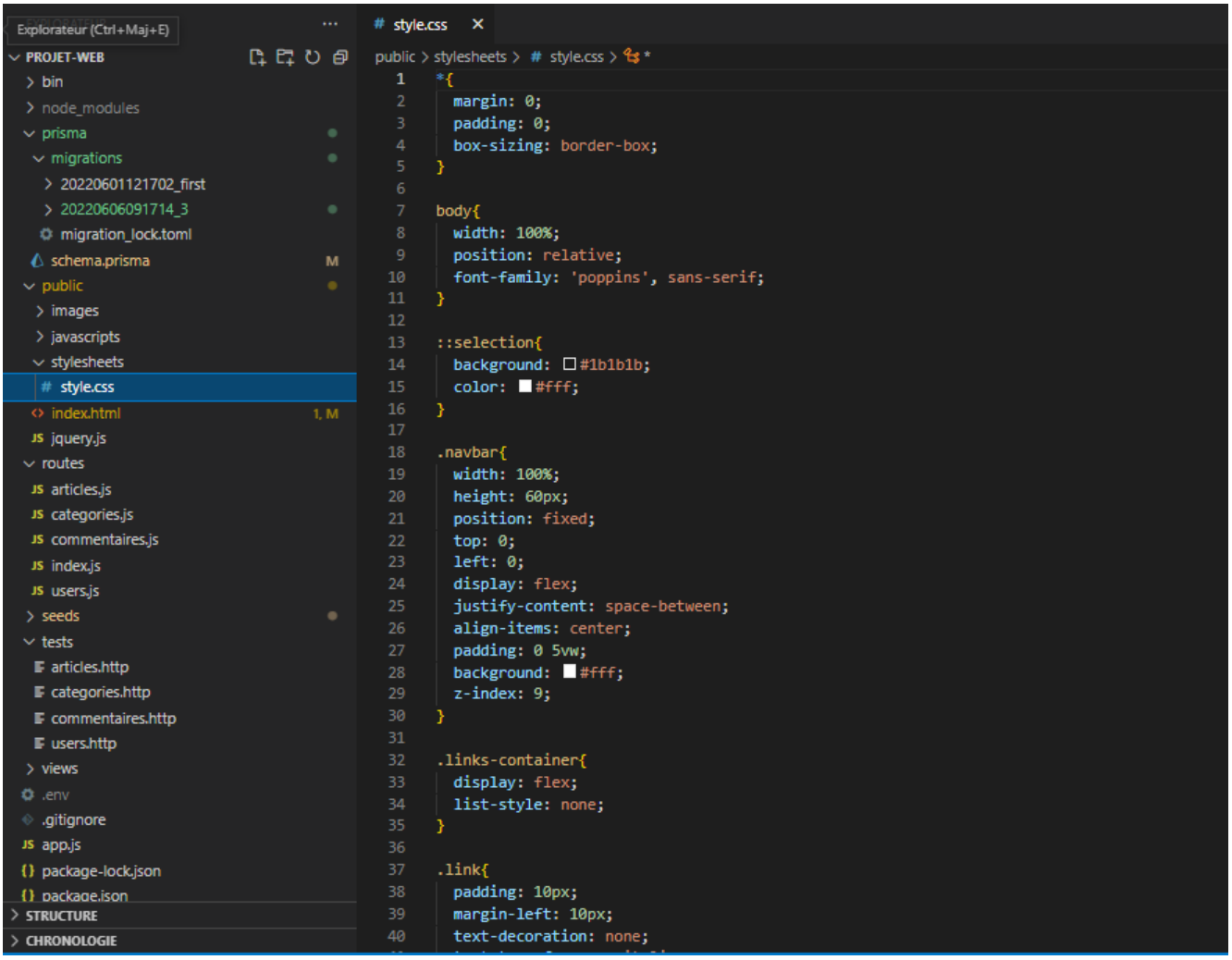



```
<!----------------------articles------------------------------>
<script>
  const url = "http://localhost:3000/articles";

  $articlesContainer = $(".container > .articles");

  function show(data) {
    const $card = $("#template > .card ");
    data.forEach((e) => {
      const $col = $('<div class = "col-md-4"></div>');
      const $clone = $card.clone(true);

      const src = $("img", $clone).attr("src");

      $("img", $clone).attr('src',e.image + '?' + e.id)
      $(".card-title", $clone).text(e.titre);
      $(".card-text", $clone).text(e.content);

      //$(".button", $clone).text(e.body);

      $col.append($clone).appendTo($articlesContainer);
    });
  }
  $.getJSON(url).then(show);
</script>
```

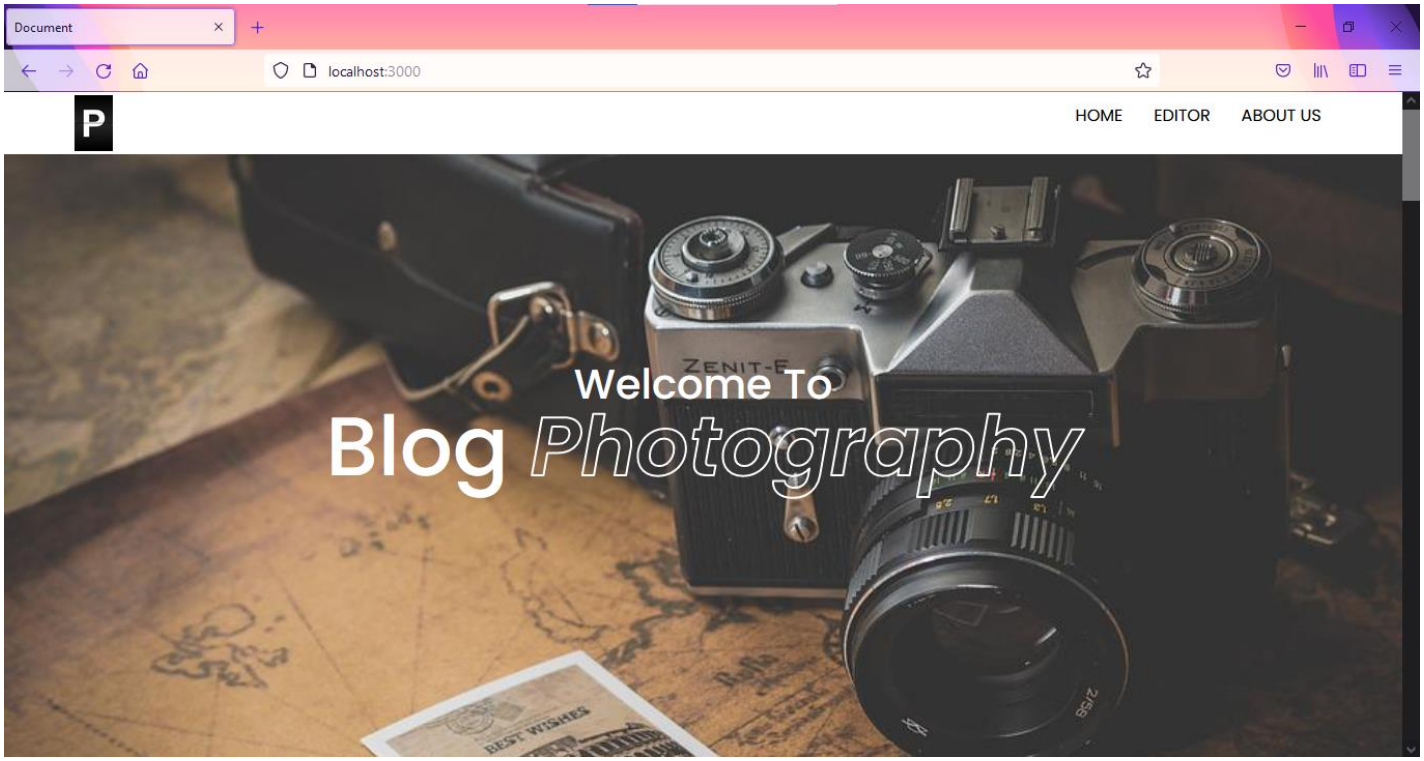
jquery.js



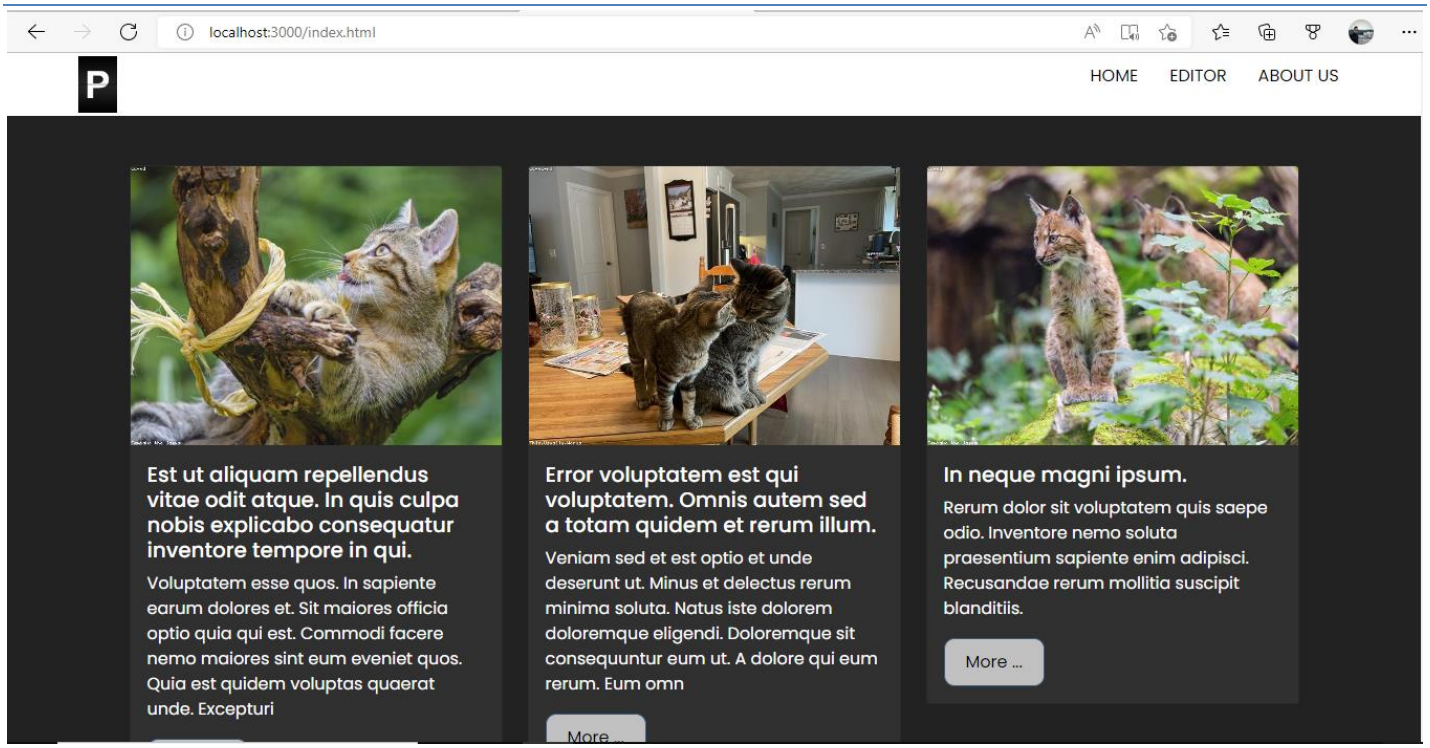
style.css



page



Les articles



CONCLUSION :

Après le passage par les différentes étapes de développement, l'application a abouti à un page web fonctionnel qui répond globalement aux critères imposés dans ce domaine.

Le présent travail nous a permis d'acquérir des connaissances dans le domaine de la programmation web, et de conforter nos connaissances.