

DÉPARTEMENT MATHÉMATIQUE INFORMATIQUE

Technologie Web

Rapport

PROJET DE FIN DE MODULE

Réaliser par :

ETOULLALI Ayoub

Professeur :

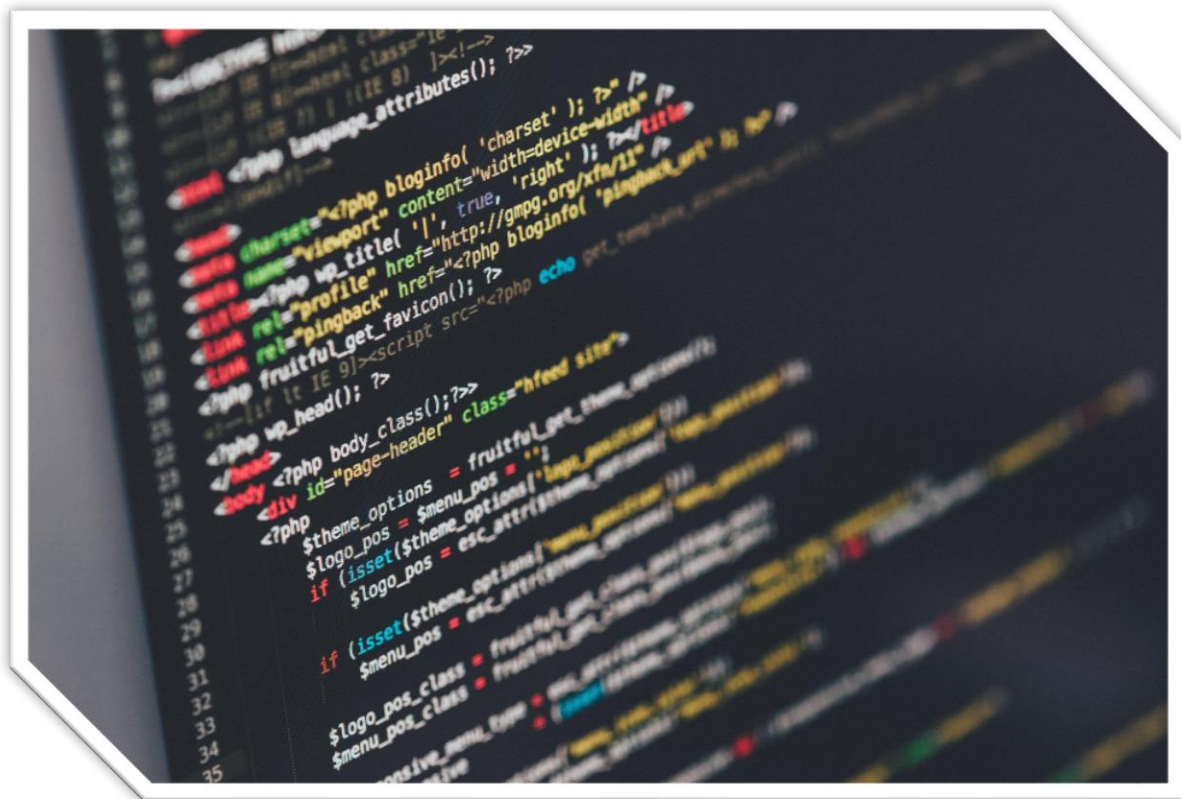
Aziz DAAIF

1ere année II-BDCC

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

SOMMAIRE

| | | |
|-------------|---------------------------|-----------|
| I. | Introduction | 02 |
| II. | Backend | 03 |
| III. | Frontend | 11 |
| IV. | Conclusion | 14 |



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.

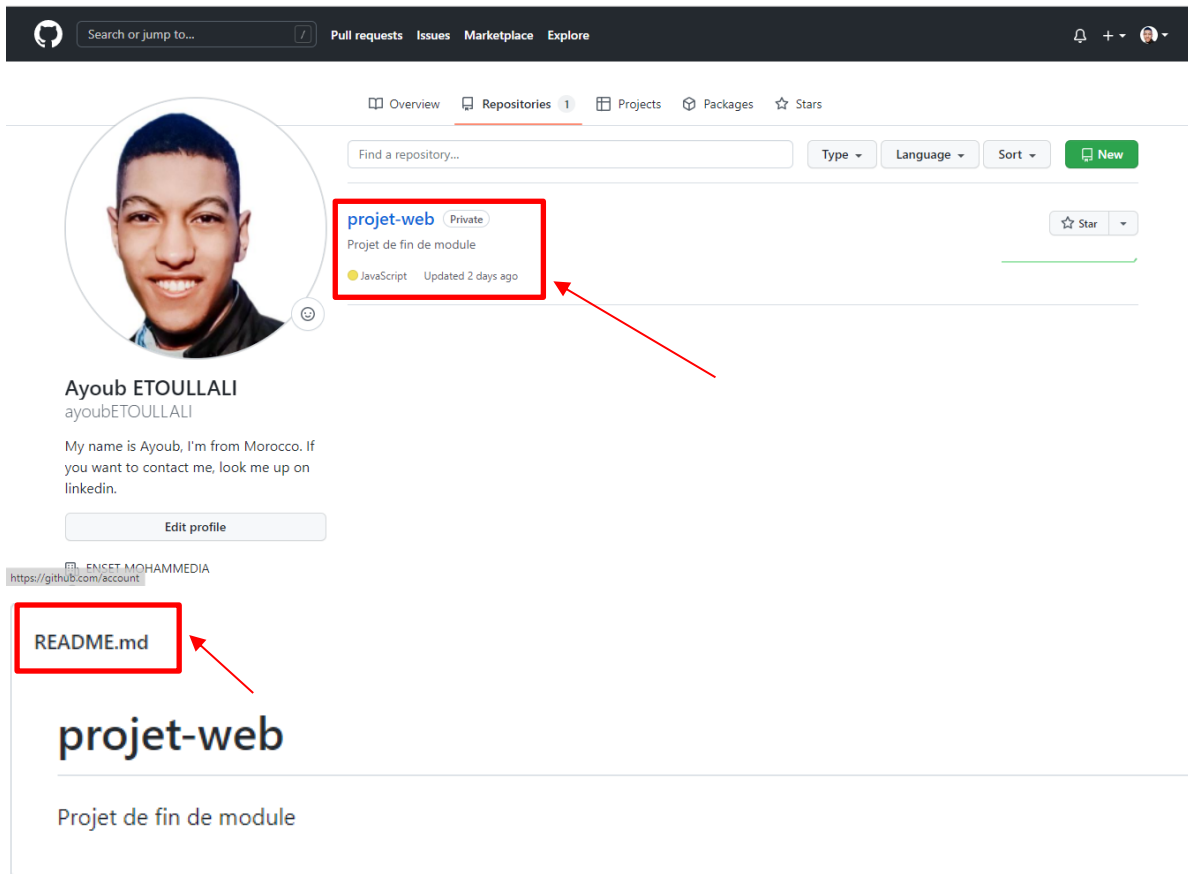
Cela peut être une application web, un site vitrine, un site e-commerce, un blog, un forum ou une application mobile.

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.

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.



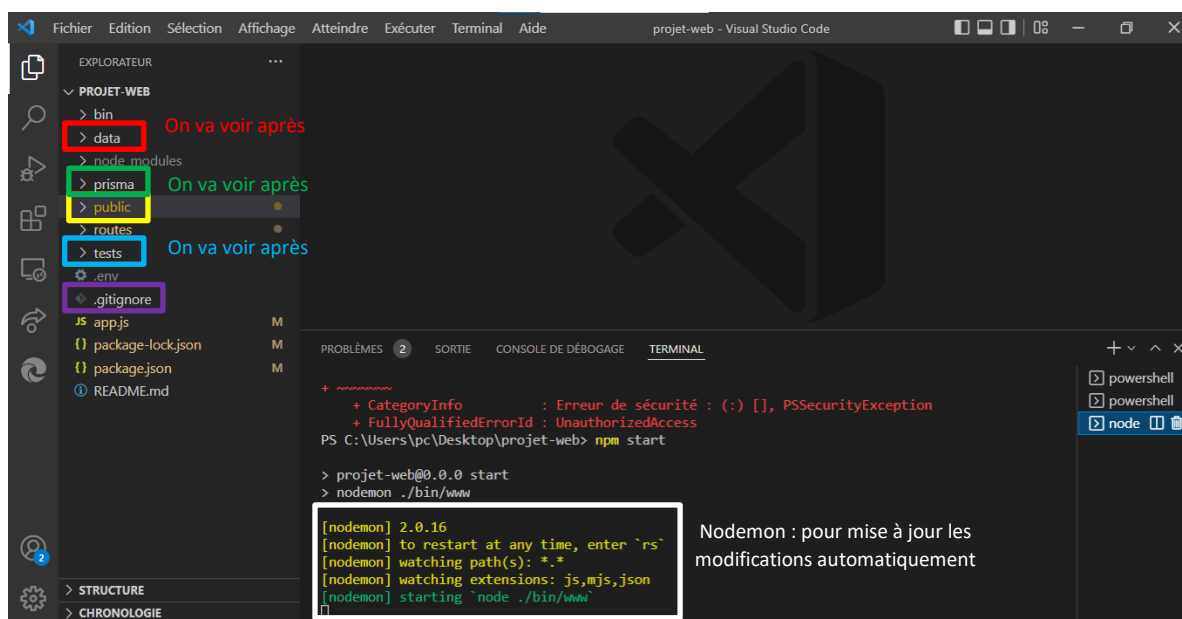
3. Initiez dans ce dossier, un projet **Express** en utilisant la commande "**express**"
Puis installez les dépendances.

Commande :

```
PS C:\Users\pc\Desktop\projet-web> npx express-generator
```

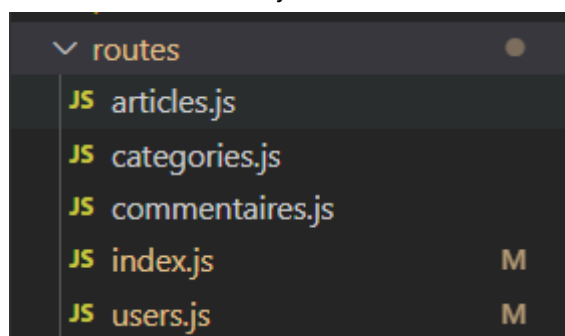
4. Ajoutez dans la racine du projet un fichier nommé **".gitignore"** contenant **"node_modules"**.

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



6. Dans le dossier routes, ajouter trois fichiers :

- articles.js
- categories.js
- commentaires.js



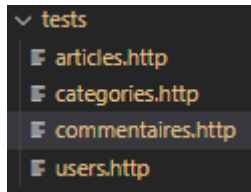
7. Pour chacun des fichiers (articles.js, categories.js, commentaires.js et users.js), ajouter les routes permettant d'effectuer les opérations de base CRUD sans prendre en considération les relations entre les entités :

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

```

1  ##
2  GET http://localhost:3000/articles
3
4  ##
5  GET http://localhost:3000/articles/5
6  Accept: application/json
7
8  ##
9
10 Send Request
11 DELETE http://localhost:3000/articles/5
12 Content-Type: application/json
13
14 ##
15 POST http://localhost:3000/articles
16 Content-Type: application/json
17
18 {
19   "titre": "",
20   "content": "",
21   "image": ""
22 }
23
24 ##
25
26 PATCH http://localhost:3000/articles/1
27 Content-Type: application/json
28
29 {
30   "titre": "hgdh",
31   "content": "hvdv",
32   "image": "hev f"
33 }

```

```

routes > JS articles.js > ...
1 var express = require('express');
2 router = express.Router();
3
4 //const articles = require('../data/articles.json')
5 const { PrismaClient } = require('@prisma/client')
6 const prisma = new PrismaClient()
7
8 /* GET articles listing. */
9
10 router.get('/', async(req, res) => {
11   const articles = await prisma.article.findMany()
12   res.json(articles)
13 });
14
15 router.get('/:id', async(req, res) => {
16   const id = +req.params.id
17   const articles = await prisma.article.findUnique({
18     where: { id },
19   })
20   res.json(articles)
21 });
22
23 router.delete('/:id', async(req, res) => {
24   const id = +req.params.id
25   const articles = await prisma.article.delete({
26     where: { id },
27   })
28   res.json(articles)
29 });
30
31 router.post('/', async(req, res) => {
32   const { titre,
33     content,
34     image } = req.body
35   const result = await prisma.article.create({
36     data: {
37       titre,
38       content,
39       image
40     }
41   })
42 }

```

```

1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 155
5 ETag: W/"9b-IEITVo8J9ACWah4cJlig9chzmU"
6 Date: Wed, 01 Jun 2022 13:02:52 GMT
7 Connection: close
8
9 [
10   {
11     "id": 1,
12     "titre": "",
13     "content": "",
14     "image": "",
15     "createdAt": "2022-06-01T12:32:54.601Z",
16     "updatedAt": "2022-06-01T12:32:54.603Z",
17     "published": false,
18     "userId": null
19   }
20 ]

```

- categories.http

```

1  ##
2  GET http://localhost:3000/categories
3
4  ##
5  GET http://localhost:3000/categories/5
6  Accept: application/json
7
8  ##
9
10 Send Request
11 DELETE http://localhost:3000/categories/5
12 Content-Type: application/json
13
14 ##
15 POST http://localhost:3000/categories
16 Content-Type: application/json
17
18 {
19   "name": ""
20 }
21
22 ##
23 PATCH http://localhost:3000/categories/1
24 Content-Type: application/json
25
26 {
27   "name": "hgdh"
28 }

```

```

routes > JS categories.js > ...
1 var express = require('express');
2 router = express.Router();
3
4 //const categories = require('../data/categories.json')
5 const { PrismaClient } = require('@prisma/client')
6 const prisma = new PrismaClient()
7
8 /* GET categories listing. */
9
10 router.get('/', async(req, res) => {
11   const categories = await prisma.categorie.findMany()
12   res.json(categories)
13 });
14
15 router.get('/:id', async(req, res) => {
16   const id = +req.params.id
17   const categorie = await prisma.categorie.findUnique({
18     where: { id },
19   })
20   res.json(categorie)
21 });
22
23 router.delete('/:id', async(req, res) => {
24   const id = +req.params.id
25   const categorie = await prisma.categorie.delete({
26     where: { id },
27   })
28   res.json(categorie)
29 });
30
31 router.post('/', async(req, res) => {
32   const { name } = req.body
33   const result = await prisma.categorie.create({
34     data: {
35       name
36     }
37   })
38   res.send(result)
39 });
40
41 router.patch('/:id', async(req, res) => {
42   const id = +req.params.id;
43 }

```

```

1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 58
5 ETag: W/"3a-PzKda24Rulda2W3QdIfg7b0XuI"
6 Date: Wed, 01 Jun 2022 13:00:51 GMT
7 Connection: close
8
9 [
10   {
11     "id": 1,
12     "name": ""
13   },
14   {
15     "id": 2,
16     "name": ""
17   },
18   {
19     "id": 3,
20     "name": ""
21   }
22 ]

```

- commentaires.http

```

tests > commentaires.http > DELETE /commentaires/5
1 ###
2 GET http://localhost:3000/commentaires
3
4 ###
5 Send Request
6 GET http://localhost:3000/commentaires/5
7 Accept: application/json
8
9 ###
10 Send Request
11 DELETE http://localhost:3000/commentaires/5
12 Content-Type: application/json
13
14 ###
15 Send Request
16 POST http://localhost:3000/commentaires
17 Content-Type: application/json
18 {
19   "email": "efa",
20   "content": "etetz"
21 }
22
23 ###
24 Send Request
25 PATCH http://localhost:3000/commentaires/1
26 Content-Type: application/json
27 {
28   "email": "pgdh",
29   "content": "hvdh"
30 }
31
routes > JS commentaires.js > router.delete('/:id') callback > where
1 var express = require('express');
2 var router = express.Router();
3
4 //const commentaires = require('../data/commentaires.json')
5 const { PrismaClient } = require('@prisma/client')
6 const prisma = new PrismaClient()
7
8 /* GET commentaires listing. */
9
10 router.get('/', async(req, res) => {
11   const commentaires = await prisma.commentaire.findMany()
12   res.json(commentaires)
13 });
14
15 router.get('/:id', async(req, res) => {
16   const id = +req.params.id
17   const commentaire = await prisma.commentaire.findUnique({
18     where: { id },
19   })
20   res.json(commentaire)
21 });
22
23 router.delete('/:id', async(req, res) => {
24   const id = +req.params.id
25   const commentaire = await prisma.commentaire.delete({
26     where: { id },
27   })
28   res.json(commentaire)
29 });
30
31 router.post('/', async(req, res) => {
32   const { email, content } = req.body
33   const result = await prisma.commentaire.create({
34     data: {
35       email,
36       content
37     },
38   })
39   res.send(result)
40 });
41
42 router.patch('/:id', async(req, res) => {
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 59
5 ETag: W/"3b-AW65x/yzBgmhMIVAGmte5zcNLS"
6 Date: Wed, 01 Jun 2022 12:57:35 GMT
7 Connection: close
8
9 [
10   {
11     "id": 1,
12     "email": "efa",
13     "content": "etetz",
14     "articleId": null
15   }
16 ]

```

- users.http

```

tests > users.http > ...
1 ###
2 Send Request
3 GET http://localhost:3000/users
4 Accept: application/json
5
6 ###
7 Send Request
8 GET http://localhost:3000/users/6
9 Accept: application/json
10
11 ###
12 Send Request
13 DELETE http://localhost:3000/users/5
14 Content-Type: application/json
15
16 ###
17 Send Request
18 POST http://localhost:3000/users
19 Content-Type: application/json
20 {
21   "email": "grdsg@gmail.com",
22   "name": "eqfzeze",
23   "role": "USER"
24 }
25
26 ###
27 Send Request
28 PATCH http://localhost:3000/users/1
29 Content-Type: application/json
30 {
31   "email": "etoulnjali@gmail.com",
32   "name": "etoulnblali",
33   "role": "USER"
34 }
35
routes > JS users.js > ...
1 var express = require('express');
2 var router = express.Router();
3
4 //const users = require('../data/users.json')
5 const { PrismaClient } = require('@prisma/client')
6 const prisma = new PrismaClient()
7
8 /* GET users listing. */
9
10 router.get('/', async(req, res) => {
11   const users = await prisma.user.findMany()
12   res.json(users)
13 });
14
15 router.get('/:id', async(req, res) => {
16   const id = +req.params.id
17   const user = await prisma.user.findUnique({
18     where: { id },
19   })
20   res.json(user)
21 });
22
23 router.delete('/:id', async(req, res) => {
24   const id = +req.params.id
25   const user = await prisma.user.delete({
26     where: { id },
27   })
28   res.json(user)
29 });
30
31 router.post('/', async(req, res) => {
32   const { email, name } = req.body
33   const result = await prisma.user.create({
34     data: {
35       email,
36       name,
37     },
38   })
39   res.send(result)
40 });
41
42 router.patch('/:id', async(req, res) => {
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

```

1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 539
5 ETag: W/"21b-B7B49T8YISjy+K/3YQtUecSgJw"
6 Date: Wed, 01 Jun 2022 12:50:13 GMT
7 Connection: close
8
9 [
10   {
11     "id": 1,
12     "email": "etoulnjali@gmail.com",
13     "name": "etoulnblali",
14     "role": "USER"
15   },
16   {
17     "id": 3,
18     "email": "etoullali@gmail.com",
19     "name": "etoullali",
20     "role": "USER"
21   },
22   {
23     "id": 4,
24     "email": "rgfr@gmail.com",
25     "name": "frg",
26     "role": "USER"
27   },
28   {
29     "id": 6,
30     "email": "said@gmail.com",
31     "name": "said",
32     "role": "USER"
33   },
34   {
35     "id": 7,
36     "email": "ayoukhhb@gmail.com",
37     "name": "ayoukhhb",
38     "role": "USER"
39   }
40 ]

```

✓ 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 SGBDR MySQL.

Assurez-vous d'avoir un serveur MySQL démarré.

```
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.

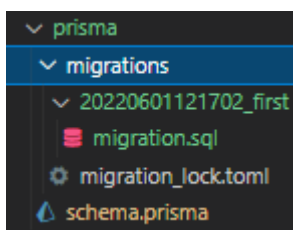
Compléter la configuration pour la prise en charge de **mysql**.

```

schema.prisma M X
prisma > schema.prisma
1 // This is your Prisma schema file,
2 // learn more about it in the docs: https://pris.ly/d/prisma-schema
3
4 generator client {
5   provider = "prisma-client-js"
6 }
7
8 datasource db {
9   provider = "mysql"
10  url      = env("DATABASE_URL")
11 }
12
13 model User {
14   id      Int      @id @default(autoincrement())
15   email   String   @unique @db.VarChar(30)
16   name    String?  @db.VarChar(20)
17   role    Role      @default(USER)
18   article Article[]
19 }
20
21 model Article {
22   id      Int      @id @default(autoincrement())
23   titre   String   @db.VarChar(255)
24   content String
25   image   String
26   createdAt DateTime @default(now())
27   updatedAt DateTime @updatedAt
28   published Boolean @default(false)
29   user    User?    @relation(fields: [userId], references: [id])
30   userId  Int?
31   categorie CategorierOnarticle[]
32   commentaire Commentaire[]
33 }
34
35 enum Role {
36   USER
37   ADMIN
38   AUTHOR
39 }
40
41 model Categorier {
42   id      Int      @id @default(autoincrement())
43   name    String?  @db.VarChar(20)
44   article CategorierOnarticle[]
45 }
46
47 model Commentaire {
48   id      Int      @id @default(autoincrement())
49   email   String   @db.VarChar(30)
50   content String   @db.VarChar(500)
51   article Article? @relation(fields: [articleId], references: [id])
52   articleId Int?
53 }
54
55 model CategorierOnarticle {
56   article      Article      @relation(fields: [articleId], references: [id])
57   articleId    Int // relation scalar field (used in the '@relation' attribute above)
58   categorie    Categorier    @relation(fields: [categorieId], references: [id])
59   categorieId  Int // relation scalar field (used in the '@relation' attribute above)
60   assignedAt  DateTime @default(now())
61   assignedBy  String
62   @@id([articleId, categorieId])
63 }
64

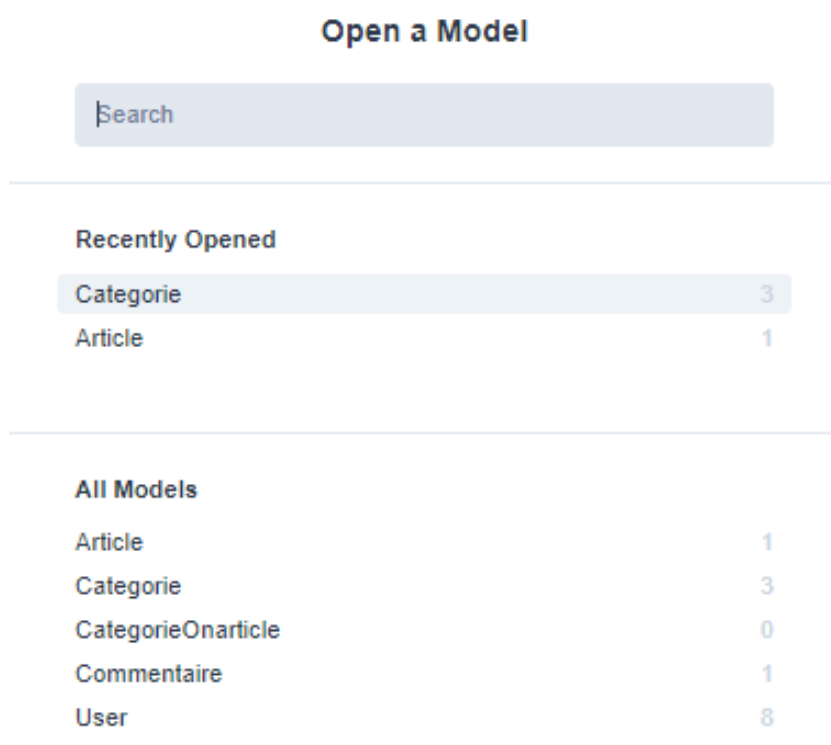
```

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



> npx prisma studio

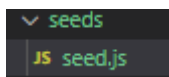
```
PS C:\Users\pc\Desktop\projet-web> npx prisma studio
Environment variables loaded from .env
Prisma schema loaded from prisma\schema.prisma
Prisma Studio is up on http://localhost:5555
```



- 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](#)

- 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
```

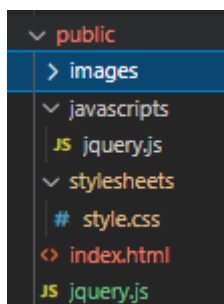
de créer en utilisant la bibliothèque “Faker” des données de tests. Vous devez créer :

- 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

```

1  const { faker } = require('@faker-js/faker')
2
3  const { PrismaClient } = require('@prisma/client')
4  const prisma = new PrismaClient()
5
6  const users = Array.from({ length: 10 }).map(() => ({
7    name: faker.lorem.firstName(),
8    email: faker.internet.email(),
9    password: faker.internet.password(),
10   role: "AUTHOR"
11  }))
12
13  const categories = Array.from({ length: 4 }).map(() => ({
14    name: faker.name.jobArea()
15  }))
16
17  const commentaires = Array.from({ length: 20 }).map(() => ({
18    email: faker.internet.email(),
19    content: faker.lorem.paragraph()
20  }))
21
22  const articles = Array.from({ length: 100 }).map(() => ({
23    titre: faker.lorem.lines(),
24    contenu: faker.lorem.paragraph(),
25    image: faker.image.Abstract(),
26    published: faker.datatype.boolean()
27  }))
28
29  async function main() {
30    await prisma.user.deleteMany();
31    await prisma.article.deleteMany();
32    await prisma.commentaire.deleteMany();
33    await prisma.categorie.deleteMany();
34
35    await prisma.user.create()
36  };
37
38  async function main() {
39    await prisma.user.deleteMany();
40    await prisma.article.deleteMany();
41    await prisma.commentaire.deleteMany();
42    await prisma.categorie.deleteMany();
  
```

Dossier public



index.html

```

EXPLORATEUR
PROJET-WEB
  > bin
  > node_modules
  > prisma
  > migrations
  > schema.prisma
  > public
    > images
    > javascripts
    > stylesheets
  > index.html 3. U
  > jquery.js U
  > routes
  > seeds
  > tests
  > views
  > .env
  > .gitignore
  > app.js M
  > package-lock.json M
  > package.json M
  > README.md

index.html 3. U X
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="UTF-8" />
5 <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7 <link
8   rel="stylesheet"
9   href="https://bootswatch.com/5/darkly/bootstrap.css"
10 />
11
12 <link href="https://fonts.googleapis.com/css2?family=Poppins:ital,wght@0,400;0,500;0,600;1,600&display=swap" rel="stylesheet">
13 <link rel="stylesheet" href="./stylesheets/style.css">
14 <title>Document</title>
15 </head>
16 <body>
17
18 <nav class="navbar">
19 
20 <ul class="links-container">
21 <li class="link-item"><a href="./index.html" class="link">HOME</a></li>
22 <li class="link-item"><a href="" class="link">EDITOR</a></li>
23 <li class="link-item"><a href="" class="link">ABOUT US</a></li>
24 </ul>
25 </nav>
26
27 <header class="header">
28 <div class="content">
29 <h1 class="heading">
30 <span class="small">welcome to</span>
31 <span class="small">blog</span>
32 <span class="no-fill">Photography </span>
33 </h1>
34 </div>
35 </header>
36
37 <!-- blog section -->
38 <section class="blogs-section">
39 <!-- <div class="blog-card">
40 

```

```

<!-- articles section -->
<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

[illegible]

style.css

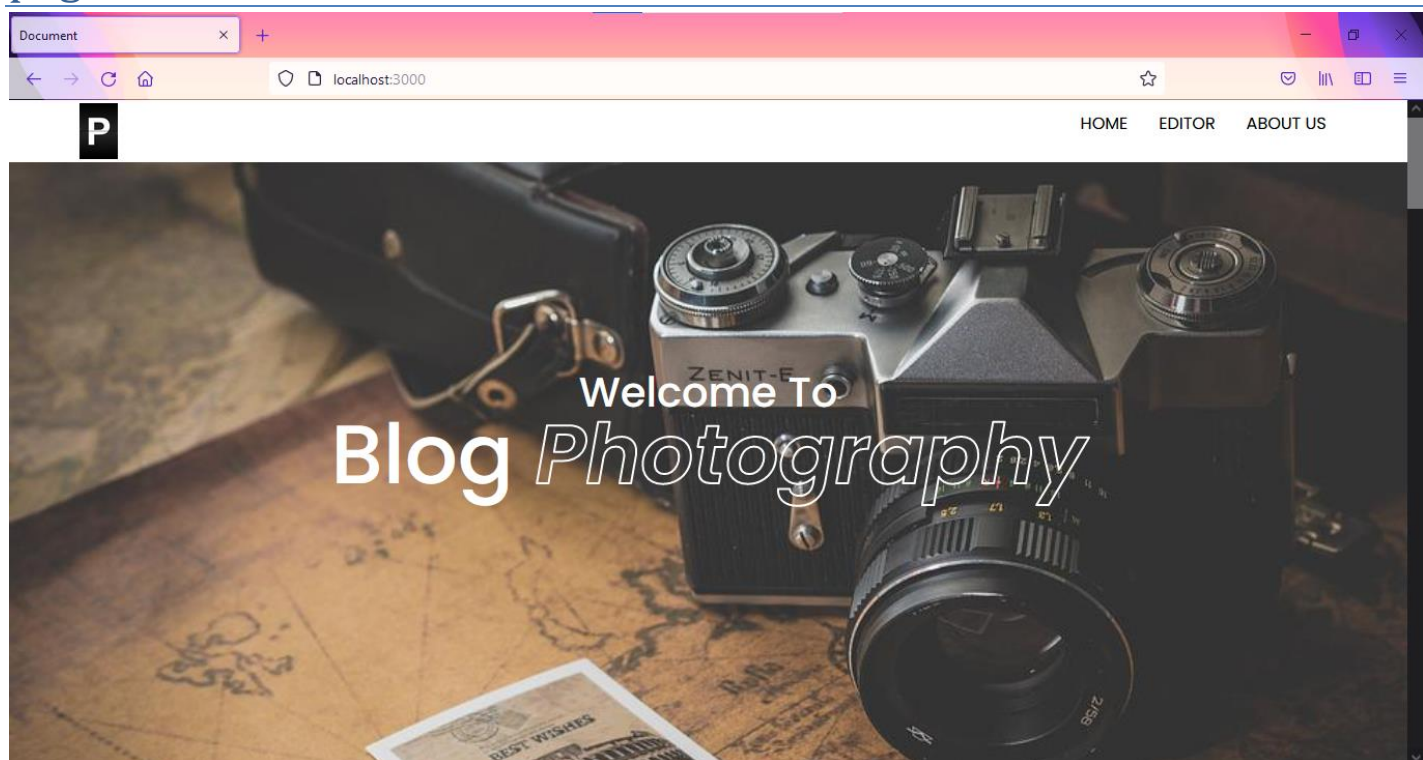
The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like bin, node_modules, prisma, migrations, public, and stylesheets. The stylesheets folder is selected, showing style.css. The style.css file contains CSS code for a navbar and links.

```

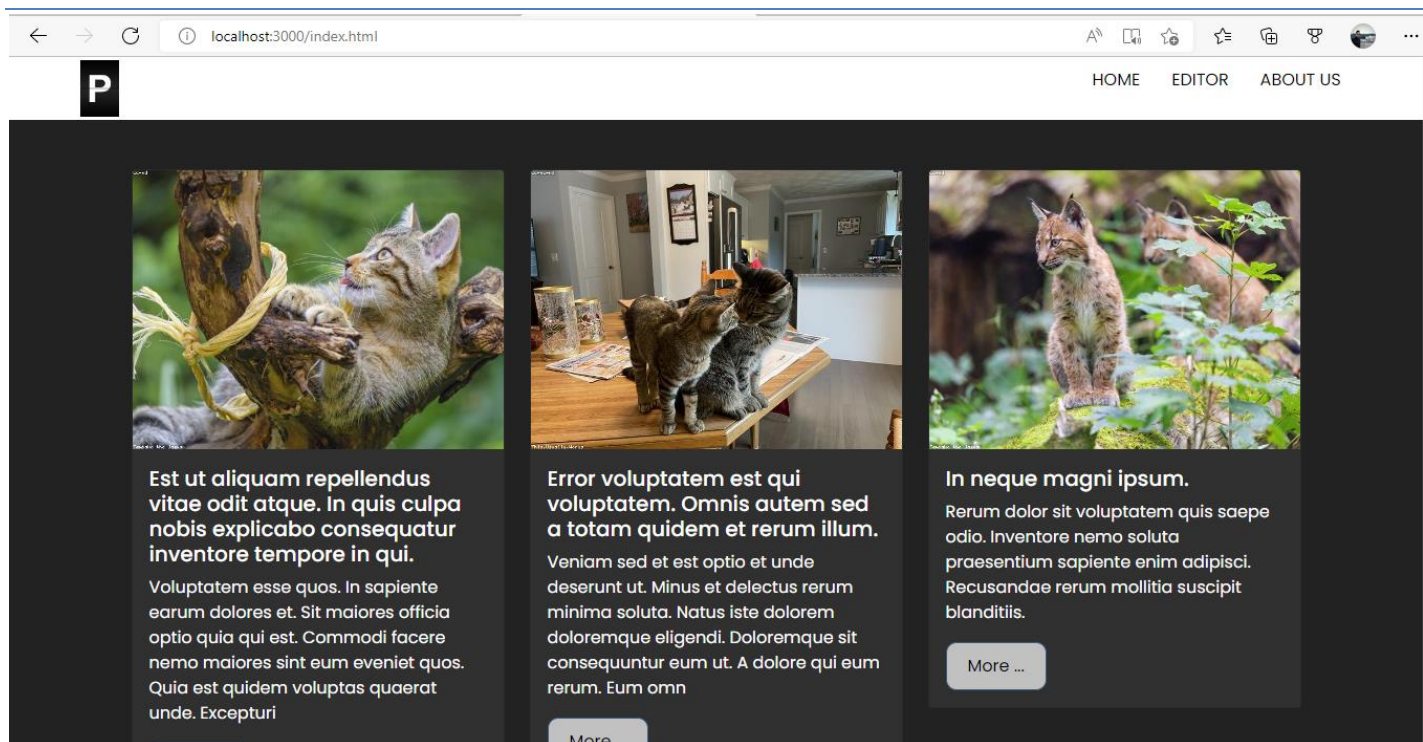
1  *{
2      margin: 0;
3      padding: 0;
4      box-sizing: border-box;
5  }
6
7  body{
8      width: 100%;
9      position: relative;
10     font-family: 'poppins', sans-serif;
11 }
12
13 ::selection{
14     background: #1b1b1b;
15     color: #fff;
16 }
17
18 .navbar{
19     width: 100%;
20     height: 60px;
21     position: fixed;
22     top: 0;
23     left: 0;
24     display: flex;
25     justify-content: space-between;
26     align-items: center;
27     padding: 0 5vw;
28     background: #fff;
29     z-index: 9;
30 }
31
32 .links-container{
33     display: flex;
34     list-style: none;
35 }
36
37 .link{
38     padding: 10px;
39     margin-left: 10px;
40     text-decoration: none;

```


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.