

RAPPORT

**Réalisation
D'un blog
Dynamique
Avec NodeJs
Express**

Réalisé par :

Achraf TAFFAH

Encadré par :

Abdelaziz DAAIF

SOMMAIRE

		INTRODUCTION
*	3	
		CAHIER DE CHARGES
*	4	
		PHASE REALISATION
*	6	
		PHASE TEST
*	22	
		PHASE DEMONSTRATION
*	26	
		CONCLUSION RÉFÉRENCE & BIBLIOGRAPHIE
*	37	

INTRODUCTION

D'abord, On utilise **Node js** pour faire des applications cross-plateforme avec des Framework comme Ionic pour les téléphones ou encore Electron pour des ordinateurs portables. Il est aussi employé parfois pour faire des serveurs web. Pour l'authentification, quelques **API Rest** sont créées avec Node. Jour après jour, de plus en plus d'applications web sont déployées pour répondre à toujours plus de besoins. Cette multiplication des projets pousse les développeurs à trouver des moyens de créer des applications plus rapidement.

Dans tous les langages de programmation, des Framework ont été créés afin d'accélérer le développement des applications web.

Les Framework sont des ensembles de composants permettant de mettre en place rapidement les fondations d'une application tout en fixant des standards aux développeurs travaillant sur le projet, ce qui facilite le travail collaboratif.

De nombreux Framework existent notamment pour le langage JavaScript, langage roi du développement web, particulièrement en vogue ces dernières années.

Pour les développeurs JavaScript – de plus en plus nombreux – qui font le choix d'utiliser le runtime **NodeJS**, un Framework se démarque comme étant une évidence pour accélérer considérablement le développement de leur projet : Express.js

CAHIER DES CHARGES

I- CONTEXTE DU PROJET

Dans ce projet je veux développer un Blog dont les parties Backend et Frontend seront développées et testées séparément. Le backend fonctionnera sous forme d'API JSON. Voici les technologies qui seront adoptées pour ce projet :

1. Du côté **backend** :

Le Framework **NodeJS Express** pour le serveur Web

L'ORM **Prisma** et une base de données **Mysql/MariaDB** pour la persistance de données

2. Du côté **frontend** :

HTML/CSS/Javascript en utilisant la bibliothèque **Javascript JQuery** et le Framework **CSS Bootstrap**

L'application **Blog** devra gérer quatre entités :

Utilisateur :

Nom,

Email,

Password,

Rôle qui doit être un parmi (ADMIN, AUTHOR)

Article

Titre,

Contenu,

Image,

createdAt, (date)

updatedAt, (date)

published (Boolean)

Catégorie

Nom

Commentaire

Email,

Contenu

Relations entre les entités :

- Un **article** est associé à **un et un seul utilisateur** (Cet utilisateur devrait avoir le rôle AUTHOR)
- Un **utilisateur** (ayant le rôle AUTHOR) peut écrire **zéro ou plusieurs articles**
- Un **article** est associé à **zéro ou plusieurs catégories**
- Une **catégorie** est associée à **zéro ou plusieurs articles**
- Un **commentaire** est associé à **exactement un article**
- Un **article** est associé à **zéro ou plusieurs commentaires**.

II - DESCRIPTION FONCTIONNELLE DES BESOINS

FONCTIONNALITÉ : L'UTILISATION DU BLOGE PAR LES UTILISATEURS.

> Objectif :

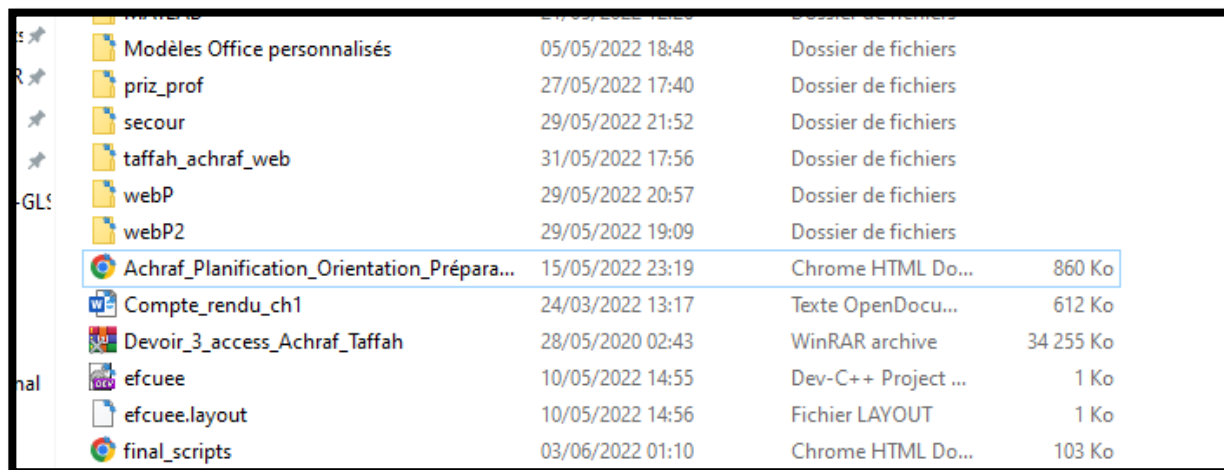
Créer des blogs et les affichées qui ont des relations avec différentes sujets ou bien catégories aux visiteurs publics.

> Description :

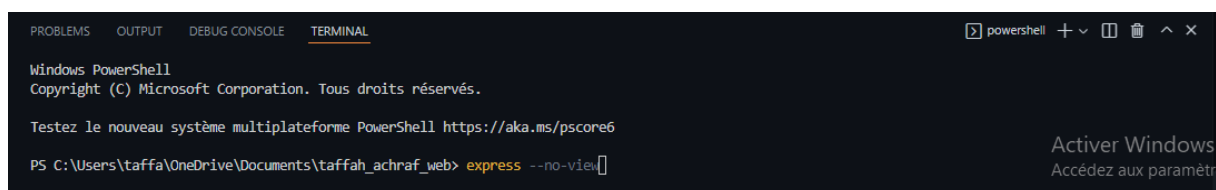
En tant qu'**Utilisateur**, si je joue un rôle d'administrateur, je peux gérer les autres utilisateurs, les articles ainsi que les catégories, si j'ai un rôle d'un simple utilisateur de peux juste consulter les blogs et ajouté des commentaires, sinon (je suis un blogueur) je peux créer des articles et les gérer, mais juste les articles que j'ai créés.

PHASE REALISATION

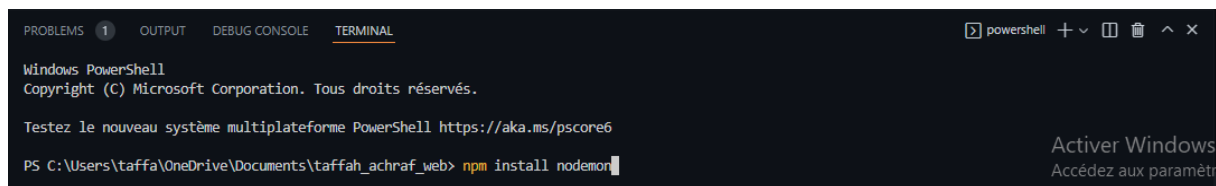
Tout d'abord, j'ai créé un dossier nommé **taffah_achraf_web**.



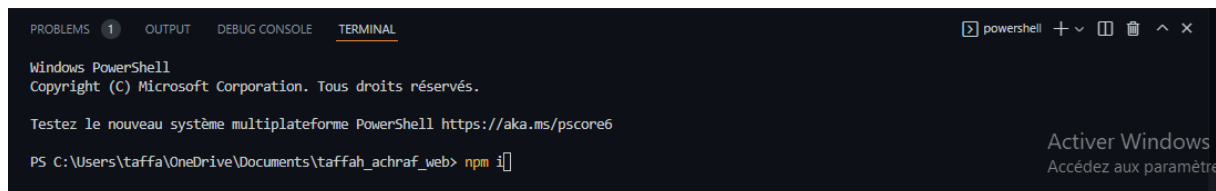
J'ai tapé la commande dans l'invite de commande **express --no-view**



Puis, j'ai installé **nodemon** avec la commande **npm i -g nodemon**.

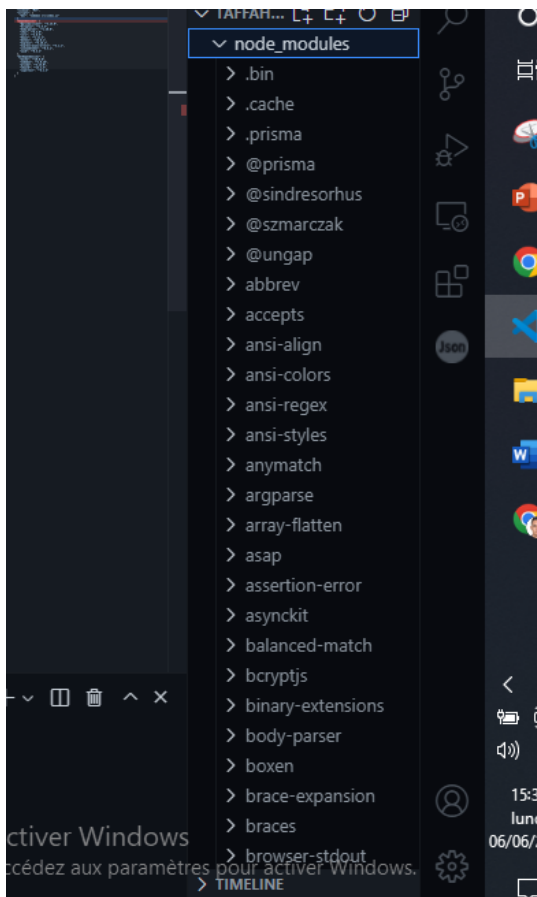


Donc maintenant je peux lancer l'exécution de mon programme avec la commande **npm run dev**.



Ensuite, j'ai tapé la commande **npm i** pour installer les **node modules**.

Voilà le dossier **node_modules** va créer dans mon projet.



Ensuite, j'ai tapé la commande **npm install prisma** pour installer l'ORM Prisma dans mon projet.

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Testez le nouveau système multiplateforme PowerShell https://aka.ms/pscore6

PS C:\Users\taffa\OneDrive\Documents\taffah_achraf_web> npm i prisma
[...]
```

J'ai tapé la commande **npm prisma init**.

```

PS C:\Users\taffa\OneDrive\Documents> npm prisma init
[...]
```

Le résultat de cette commande.

```

src > prisma > JS client.js > ...
1  const {PrismaClient} = require("@prisma/client");
2
3  const client = new PrismaClient();
4
5  module.exports = { client };
6
```

Cette commande a affiché répertoire un fichier un fichier nommé **.env**.



Ensuite, j'ai réalisé la création de ma base de données nommé **taffah_blog**.

Puis j'ai rempli ce fichier avec les informations suivantes qui concernent ma base de données, le nom d'utilisateur, le nom de la base de données ainsi que le mot de passe.

```
.env
1
2 DATABASE_URL="mysql://root:@localhost:3306/taffah_blog"
```

Maintenant, j'ai essayé de créer les tables de ma base de données avec le script de l'ORM **prisma**.

```
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 Utilisateur {
14   id      String @id @default(uuid())
15   nom     String @unique
16   email   String @unique
17   password String
18   role    Role   @default(AUTHOR)
19
20   Articles Article[]
21 }
22
23 model Article {
24
25   id      Int    @id @default(autoincrement())
26   titre   String
27   contenu String @db.LongText
28   image   String
29   createdAt DateTime @default(now())
30   updatedAt DateTime @default(now())
31   published Boolean @default(false)
32
33   authorId String
34   author   Utilisateur @relation(fields: [authorId], references: [id])
35 }
```

```

23 model Article {
24   id      Int      @id @default(autoincrement())
25   titre   String
26   contenu String    @db.LongText
27   image   String
28   createdAt DateTime @default(now())
29   updatedAt DateTime @default(now())
30   published Boolean   @default(false)
31
32   authorId String
33   author  Utilisateur @relation(fields: [authorId], references: [id])
34
35   categories CategoriesOnArticles[]
36   commentaire Commentaire[]
37 }
38
39
40 model Categorie {
41   id      Int      @id @default(autoincrement())
42   nom     String
43   Articles CategoriesOnArticles[]
44 }
45
46 model CategoriesOnArticles {
47   articleId Int
48   categorieId Int
49
50   article  Article @relation(fields: [articleId], references: [id])
51   categorie Categorie @relation(fields: [categorieId], references: [id])
52
53   @@id([articleId, categorieId])
54 }
55
56

```

```

model Commentaire {
  id      Int      @id @default(autoincrement())
  email   String
  contenu String    @db.LongText
  articleId Int
  article  Article @relation(fields: [articleId], references: [id])
}

enum Role {
  ADMIN
  AUTHOR
  USER
}

```

Maintenant pour créer ces tables dans ma base de données, il est nécessaire de taper ce commande **npx prisma migrate dev**.

```
PS C:\Users\taffa\OneDrive\Documents\taffah_achraf_web> npx prisma migrate dev
```

Voilà maintenant le résultat de cette commande.

La création et l'insertion de ces tables dans la base de données.

```

1  -- CreateTable
2  CREATE TABLE `Utilisateur` (
3      `id` VARCHAR(191) NOT NULL,
4      `nom` VARCHAR(191) NOT NULL,
5      `email` VARCHAR(191) NOT NULL,
6      `password` VARCHAR(191) NOT NULL,
7      `role` ENUM('ADMIN', 'AUTHOR') NOT NULL DEFAULT 'AUTHOR',
8
9      UNIQUE INDEX `Utilisateur_nom_key`(`nom`),
10     UNIQUE INDEX `Utilisateur_email_key`(`email`),
11     PRIMARY KEY (`id`)
12 ) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
13
14  -- CreateTable
15  CREATE TABLE `Article` (
16      `id` INTEGER NOT NULL AUTO_INCREMENT,
17      `titre` VARCHAR(191) NOT NULL,
18      `contenu` LONGTEXT NOT NULL,
19      `image` VARCHAR(191) NOT NULL,
20      `createdAt` DATETIME(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
21      `updatedAt` DATETIME(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3),
22      `published` BOOLEAN NOT NULL DEFAULT false,
23      `authorId` VARCHAR(191) NOT NULL,
24
25      PRIMARY KEY (`id`)
26 ) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```

-- CreateTable
CREATE TABLE `Categorie` (
    `id` INTEGER NOT NULL AUTO_INCREMENT,
    `nom` VARCHAR(191) NOT NULL,

    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- CreateTable
CREATE TABLE `CategoriesOnArticles` (
    `articleId` INTEGER NOT NULL,
    `categorieId` INTEGER NOT NULL,

    PRIMARY KEY (`articleId`, `categorieId`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- CreateTable
CREATE TABLE `Commentaire` (
    `id` INTEGER NOT NULL AUTO_INCREMENT,
    `email` VARCHAR(191) NOT NULL,
    `contenu` LONGTEXT NOT NULL,
    `articleId` INTEGER NOT NULL,

    PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

```

```

CREATE TABLE `RefreshToken` (
  `id` VARCHAR(191) NOT NULL,
  `expiredIn` INTEGER NOT NULL,
  `userId` VARCHAR(191) NOT NULL,

  UNIQUE INDEX `RefreshToken_userId_key` (`userId`),
  PRIMARY KEY (`id`)
) DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

-- AddForeignKey
ALTER TABLE `Article` ADD CONSTRAINT `Article_authorId_fkey` FOREIGN KEY (`authorId`) REFERENCES `Utilisateur` (`id`) ON DELETE RESTRICT

-- AddForeignKey
ALTER TABLE `CategoriesOnArticles` ADD CONSTRAINT `CategoriesOnArticles_articleId_fkey` FOREIGN KEY (`articleId`) REFERENCES `Article` (`id`) ON DELETE RESTRICT

-- AddForeignKey
ALTER TABLE `CategoriesOnArticles` ADD CONSTRAINT `CategoriesOnArticles_categorieId_fkey` FOREIGN KEY (`categorieId`) REFERENCES `Categorie` (`id`) ON DELETE RESTRICT

-- AddForeignKey
ALTER TABLE `Commentaire` ADD CONSTRAINT `Commentaire_articleId_fkey` FOREIGN KEY (`articleId`) REFERENCES `Article` (`id`) ON DELETE RESTRICT

-- AddForeignKey
ALTER TABLE `RefreshToken` ADD CONSTRAINT `RefreshToken_userId_fkey` FOREIGN KEY (`userId`) REFERENCES `Utilisateur` (`id`) ON DELETE RESTRICT

```

Donc après une modification de script qui existent dans le fichier **prisma/shema.prisma**, j'ai tapé une autre fois la commande **npx prisma migrate dev**, voilà le résultat de cette modification.

```

2 Warnings:
3
4 - You are about to drop the `refreshtoken` table. If the table is not empty, all the data it contains will be lost.
5
6 */
7 -- DropForeignKey
8 ALTER TABLE `refreshtoken` DROP FOREIGN KEY `RefreshToken_userId_fkey`;
9
10 -- AlterTable
11 ALTER TABLE `utilisateur` MODIFY `role` ENUM('ADMIN', 'AUTHOR', 'USER') NOT NULL DEFAULT 'AUTHOR';
12
13 -- DropTable
14 DROP TABLE `refreshtoken`;
15

```

Donc tous les tables avec la relation ont créé dans la base de données **taffah_blog**.

Table	Action	Lignes	Type	Interclassement	Taille	Perte
article	Afficher Structure Rechercher Insérer Vider Supprimer	~17	InnoDB	utf8mb4_unicode_ci	32 Kio	-
categorie	Afficher Structure Rechercher Insérer Vider Supprimer	~6	InnoDB	utf8mb4_unicode_ci	16 Kio	-
categoriesonarticles	Afficher Structure Rechercher Insérer Vider Supprimer	~8	InnoDB	utf8mb4_unicode_ci	32 Kio	-
commentaire	Afficher Structure Rechercher Insérer Vider Supprimer	~3	InnoDB	utf8mb4_unicode_ci	32 Kio	-
utilisateur	Afficher Structure Rechercher Insérer Vider Supprimer	~5	InnoDB	utf8mb4_unicode_ci	48 Kio	-
_prisma_migrations	Afficher Structure Rechercher Insérer Vider Supprimer	~2	InnoDB	utf8mb4_unicode_ci	16 Kio	-
6 tables	Somme	33	InnoDB	latin1_swedish_ci	176 Kio	0 0

Maintenant, j'ai créé un dossier nommé qui contient les dossiers suivants :

Public : ici je veux créer le code de la partie **front-end**.

Routes : ici je veux créer le code qui permet de faire circuler entre les pages à travers les adresses URL.

UseCases : ici je veux créer les contrôleurs pour assumer le fonctionnement des requêtes de création, modification, lecture ainsi que de la suppression.

Exemple : la gestion des articles

La ligne dans lequel j'ai écrit **Res.header(« Access-Control-Allow-Origin », « * »)** donne la possibilité à n'importe quel client de consommer notre **Api**.

```
const { CreateArticleUseCase } = require("../CreateArticleUseCase");

class CreateArticleController {
  async handle(req, res) {
    res.header("Access-Control-Allow-Origin", "*");
    const { titre, contenu, image, authorId, published } = req.body;
    const article = {
      titre: titre,
      contenu: contenu,
      image: image,
      published: published,
      authorId: authorId
    };
    const createArticleUseCase = new CreateArticleUseCase();
    const createdArticle = await createArticleUseCase.execute(article);
    res.json({
      article: createdArticle,
    });
  }
}

module.exports = { CreateArticleController };
```

Voilà comment peut-on créer des articles en faisant appelle à notre ORM Prisma, la commande `ma` permet d'inclure le fichier `./prisma/client.js`

```
const { client } = require("../prisma/client");

class CreateArticleUseCase {
  async execute(article) {
    const Createdarticle = await client.article.create({
      data: {
        ...article,
      },
    });
    return Createdarticle;
  }
}

module.exports = { CreateArticleUseCase };
```

Le code qui permet de la suppression d'un article.

```

const { DeleteArticleUseCase } = require("../DeleteArticleUseCase");

class DeleteArticleController {
  async handle(req, res) {
    res.header("Access-Control-Allow-Origin", "*");
    const { id } = req.params;
    const deleteArticleUseCase = new DeleteArticleUseCase();
    try {
      await deleteArticleUseCase.execute(parseInt(id));
    } catch (error) {
      throw new Error("Article Doesn't Exists");
    }
    res.json({
      status: "DELETED",
      message: "Article: " + id,
    });
  }
}

module.exports = { DeleteArticleController };

```

Ce code me permet de supprimer toutes les relations avec l'article qui je veux la supprimé.

```

1  const { client } = require("../prisma/client");
2
3  class DeleteArticleUseCase {
4    async execute(articleId) {
5      await client.commentaire.deleteMany({
6        where: {
7          articleId,
8        },
9      });
10     await client.categoriesOnArticles.deleteMany({
11       where: {
12         articleId,
13       },
14     });
15     await client.article.delete({
16       where: {
17         id: articleId,
18       },
19     });
20   }
21 }
22 module.exports = { DeleteArticleUseCase };
23

```

Maintenant pour extraire tous les articles j'ai utilisé le code suivant.

```

const { GetAllArticlesUseCase } = require("../GetAllArticlesUseCase");

class GetAllArticlesController {
  async handle(req, res) {
    res.header("Access-Control-Allow-Origin", "*");
    const { skip, take } = req.query;
    const getAllArticlesUseCase = new GetAllArticlesUseCase();

    const skipping = skip && parseInt(skip) > 0 ? parseInt(skip) : 0;
    const taking = take && parseInt(take) > 0 ? parseInt(take) : 0;

    var articles = await getAllArticlesUseCase.execute();

    if (skipping)
      articles = articles.slice(skipping, skipping + articles.length);
    if (taking) articles = articles.slice(0, taking);

    res.json({ articles });
  }
}
module.exports = { GetAllArticlesController };

```

```

rc > useCases > articles > getAllArticles > JS GetAllArticlesUseCase.js > ...
1  const { client } = require("../../prisma/client");
2
3  class GetAllArticlesUseCase {
4    async execute() {
5      const articles = await client.article.findMany({
6        include: {
7          categories: {
8            include: {
9              categorie: {
10               select: {
11                 nom: true,
12               },
13             },
14           },
15         },
16       });
17     });
18     return articles;
19   }
20 }
21 module.exports = { GetAllArticlesUseCase };
22

```

Pour extraire un article précis à travers un identifiant j'ai utilisé le code suivant.

```

c > useCases > articles > getArticle > JS GetArticleController.js > ...
1  const { GetArticleUseCase } = require("../GetArticleUseCase");
2
3  class GetArticleController {
4    async handle(req, res) {
5      res.header("Access-Control-Allow-Origin", "*");
6      const { id } = req.params;
7      const getArticleUseCase = new GetArticleUseCase();
8      const article = await getArticleUseCase.execute(parseInt(id));
9      res.json({ article });
10   }
11 }
12 module.exports = { GetArticleController };
13

```

```

1  const { client } = require("../../../../../prisma/client");
2
3  class GetArticleUseCase {
4    async execute(id) {
5      const article = await client.article.findFirst({
6        where: {
7          id,
8        },
9        include: {
10          categories: {
11            include: {
12              categorie: {
13                select: {
14                  nom: true,
15                },
16              },
17            },
18          },
19        },
20      });
21      return article;
22    }
23  }
24  module.exports = { GetArticleUseCase };
25

```


Concernant la modification, j'ai utilisé le même principe.

```
c > useCases > articles > updateArticle > JS UpdateArticleController.js > UpdateArticleController > handle > article
1  const dayjs = require("dayjs");
2  const { UpdateArticleUseCase } = require("../UpdateArticleUseCase");
3
4  class UpdateArticleController {
5    async handle(req, res) {
6      res.header("Access-Control-Allow-Origin", "");
7      const { id, titre, contenu, image, authorId, published } = req.body;
8      const article = {
9        id: id,
10       titre: titre,
11       contenu: contenu,
12       image: image,
13       updatedAt: dayjs().toDate(),
14       published: published,
15       authorId: authorId,
16     };
17     const updateArticleUseCase = new UpdateArticleUseCase();
18     var updatedArticle = undefined;
19     try {
20       updatedArticle = await updateArticleUseCase.execute({ article });
21     } catch (error) {
22       throw new Error("Article Doesn't Exists");
23     }
24     res.json({
25       status: "UPDATED",
26       message: "Article " + id + " updated successfully",
27       article: updatedArticle,
28     });
29   }
30 }
31 module.exports = { UpdateArticleController };
32
```

```
src > useCases > articles > updateArticle > JS UpdateArticleUseCase.js > ...
1  const { client } = require("../../prisma/client");
2
3  class UpdateArticleUseCase {
4    async execute({ article }) {
5      const updatedArticle = await client.article.update({
6        where: {
7          id: article.id,
8        },
9        data: {
10         titre: article.titre,
11         contenu: article.contenu,
12         image: article.image,
13         updatedAt: article.updatedAt,
14         published: article.published,
15         authorId: article.authorId,
16       },
17     });
18     return updatedArticle;
19   }
20 }
21 module.exports = { UpdateArticleUseCase };
22
```

Remarque :

J'ai utilisé le même principe pour la gestion, des utilisateurs, des commentaires, des catégories, ainsi que l'authentification.

Dans le dossier Routes j'ai créé les fichiers suivants :

Dans le fichier Articles.js j'ai créé ce code qui va me permet d'utilisé les requêtes http, comme « **Post,Get,Patch,Delete** » en ce qui concerne les articles.

```

src > routes > JS articles.js > ...
1  const {
2    CreateArticleController,
3  } = require("../useCases/articles/createArticle/CreateArticleController");
4  const {
5    DeleteArticleController,
6  } = require("../useCases/articles/deleteArticle/DeleteArticleController");
7  const {
8    GetAllArticlesController,
9  } = require("../useCases/articles/getAllArticles/GetAllArticlesController");
10 const {
11   GetArticleController,
12 } = require("../useCases/articles/getArticle/GetArticleController");
13 const {
14   UpdateArticleController,
15 } = require("../useCases/articles/updateArticle/UpdateArticleController");
16 const {
17   AddArticleToCategorieController,
18 } = require("../useCases/categorieOnArticle/addArticleToCategorie/addArticleToCategorieController");
19 const {
20   DeleteArticleFromCategorieController,
21 } = require("../useCases/categorieOnArticle/deleteArticleFromCategorie/deleteArticleFromCategorieController");
22
23 const articlesRouter = require("express").Router();
24 const createArticleController = new CreateArticleController();
25 const getAllArticlesController = new GetAllArticlesController();
26 const getArticleController = new GetArticleController();
27 const updateArticleController = new UpdateArticleController();
28 const deleteArticleController = new DeleteArticleController();
29
30 const addArticleToCategorieController = new AddArticleToCategorieController();
31 const deleteArticleFromCategorieController =
32   new DeleteArticleFromCategorieController();

```

```

articlesRouter.get("/", getAllArticlesController.handle);
articlesRouter.get("/:id", getArticleController.handle);
articlesRouter.patch("/", updateArticleController.handle);
articlesRouter.post("/", createArticleController.handle);
articlesRouter.delete(
  "/:id",
  deleteArticleController.handle
);

articlesRouter.post(
  "/addCategorie",
  addArticleToCategorieController.handle
);
articlesRouter.post(
  "/deleteCategorie",
  deleteArticleFromCategorieController.handle
);

module.exports = { articlesRouter };

```

Dans le fichier **auth.js** j'ai créé ce code qui va m'aider pour vérifier si l'authentification a été bien faite ou bien avant de consulter les données de l'Api.

```

src > routes > JS auth.js > ...
1  const authRouter = require("express").Router();
2  const {
3    AuthenticateUserController,
4  } = require("../useCases/authentication/authenticateUser/AuthenticateUserController");
5
6  const authenticateUserController = new AuthenticateUserController();
7
8  authRouter.post("/login", authenticateUserController.handle);
9
10 module.exports = { authRouter };
11

```

Dans le fichier Articles.js j'ai créé ce code qui va me permet d'utilisé les requêtes http, comme « **Post,Get,Patch,Delete** » en ce qui concerne les catégories des articles.

```

1  const {
2    CreateCategorieController,
3  } = require("../useCases/categories/createCategorie/CreateCategorieController");
4  const {
5    DeleteCategorieController,
6  } = require("../useCases/categories/deleteCategorie/DeleteCategorieController");
7  const {
8    GetAllCategoriesController,
9  } = require("../useCases/categories/getAllCategories/GetAllCategoriesController");
10 const {
11   GetCategorieController,
12 } = require("../useCases/categories/getCategorie/GetCategorieController");
13 const {
14   UpdateCategorieController,
15 } = require("../useCases/categories/updateCategorie/UpdateCategorieController");
16
17 const categoriesRouter = require("express").Router();
18
19 const getAllCategoriesController = new GetAllCategoriesController();
20 const getCategorieController = new GetCategorieController();
21 const updateCategorieController = new UpdateCategorieController();
22 const deleteCategorieController = new DeleteCategorieController();
23 const createCategorieController = new CreateCategorieController();
24

```

```

categoriesRouter.get("/", getAllCategoriesController.handle);
categoriesRouter.get("/:id", getCategorieController.handle);
categoriesRouter.patch(
  "/",
  updateCategorieController.handle
);
categoriesRouter.post(
  "/",
  createCategorieController.handle
);
categoriesRouter.delete(
 ("/:id",
  deleteCategorieController.handle
);

module.exports = { categoriesRouter };

```

Dans le fichier Commentaire.js j'ai créé ce code qui va me permet d'utilisé les requêtes http, comme « **Post,Get,Patch,Delete** » en ce qui concerne les catégories des commentaires.

```

1 routes > JS commentaires.js > ...
2 const {
3   CreateCommentaireController,
4 } = require("../useCases/commentaires/createCommentaire/CreateCommentaireController");
5 const {
6   DeleteCommentaireController,
7 } = require("../useCases/commentaires/deleteCommentaire/DeleteCommentaireController");
8 const {
9   GetAllCommentairesController,
10 } = require("../useCases/commentaires/getAllCommentaires/GetAllCommentairesController");
11 const {
12   GetCommentaireController,
13 } = require("../useCases/commentaires/getCommentaire/GetCommentaireController");
14 const {
15   UpdateCommentaireController,
16 } = require("../useCases/commentaires/updateCommentaire/UpdateCommentaireController");
17
18 const commentairesRouter = require("express").Router();
19
20 const getAllCommentairesController = new GetAllCommentairesController();
21 const getCommentaireController = new GetCommentaireController();
22 const updateCommentaireController = new UpdateCommentaireController();
23 const deleteCommentaireController = new DeleteCommentaireController();
24 const createCommentaireController = new CreateCommentaireController();
25
26 commentairesRouter.get("/", getAllCommentairesController.handle);
27 commentairesRouter.get("/:id", getCommentaireController.handle);
28 commentairesRouter.patch(
29   "/",
30   updateCommentaireController.handle
31 );
32 commentairesRouter.post("/", createCommentaireController.handle);
33 commentairesRouter.delete(
34   "/",
35   deleteCommentaireController.handle
36 );

```

Dans le fichier users.js j'ai créé ce code qui va me permet d'utilisé les requêtes http, comme « **Post,Get,Patch,Delete** » en ce qui concerne les catégories des utilisateurs.

```

1 routes > JS users.js > ...
2 const usersRouter = require("express").Router();
3
4 const {
5   CreateUserController,
6 } = require("../useCases/users/createUser/CreateUserController");
7 const {
8   DeleteUserController,
9 } = require("../useCases/users/deleteUser/DeleteUserController");
10 const {
11   GetAllUsersController,
12 } = require("../useCases/users/getAllUsers/GetAllUsersController");
13 const {
14   GetUserController,
15 } = require("../useCases/users/getUser/GetUserController");
16 const {
17   UpdateUserController,
18 } = require("../useCases/users/updateUser/updateUserController");
19
20 const createUserController = new CreateUserController();
21 const updateUserController = new UpdateUserController();
22 const getAllUsersController = new GetAllUsersController();
23 const getUserController = new GetUserController();
24 const deleteUserController = new DeleteUserController();
25
26 usersRouter.post("/", createUserController.handle);
27 usersRouter.patch("/", updateUserController.handle);
28 usersRouter.delete("/:id", deleteUserController.handle);
29 usersRouter.get("/", getAllUsersController.handle);
30 usersRouter.get("/:id", getUserController.handle);
31
32 module.exports = { usersRouter };
33

```

Dans le fichier index.js j'ai appelé tous les fichiers que j'ai été déjà réalisé, **article.js**, **auth.js**, **categories.js**, **commentaires.js** ainsi que **users.js**.

```
src > routes > JS index.js > ...
1  const { articlesRouter } = require("./articles");
2  const { authRouter } = require("./auth");
3  const { categoriesRouter } = require("./categories");
4  const { commentairesRouter } = require("./commentaires");
5  const { usersRouter } = require("./users");
6
7  module.exports = function (app) {
8    app.use("/auth", authRouter);
9    app.use("/article", articlesRouter);
10   app.use("/commentaire", commentairesRouter);
11   app.use("/categorie", categoriesRouter);
12   app.use("/users", usersRouter);
13 };
14
```

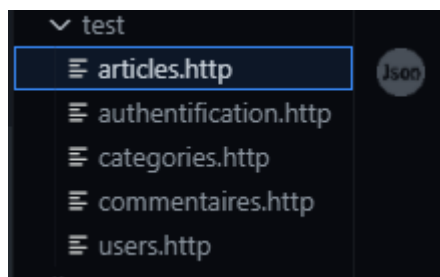
Pour la gestion des erreurs des exception, j'ai créé ce fichier dans lequel je vais le rappeler à chaque dans mon projet.

```
src > JS errorHandling.js > [e] errorHandling
1  const errorHandling = (err, req, res, next) => {
2    var status = undefined;
3    if (err.message.includes("Already Exists")) res.status(409);
4    if (err.message.includes("Already belong to")) res.status(409);
5    if (err.message.includes("Doesn't belong to")) res.status(409);
6
7    if (err.message.includes("not found")) {
8      status = "Not Found";
9      res.status(404);
10   }
11   if (err.message.includes("Doesn't Exists")) {
12     status = "Not Found";
13     res.status(404);
14   }
15
16   if (err.message.includes("incorrect")) res.status(401);
17   if (err.message.includes("invalid")) res.status(401);
18   if (err.message.includes("is missing")) res.status(401);
19
20   return res.json({ status: status ? status : "Error", message: err.message });
21 };
22
23 module.exports = { errorHandling };
24
```

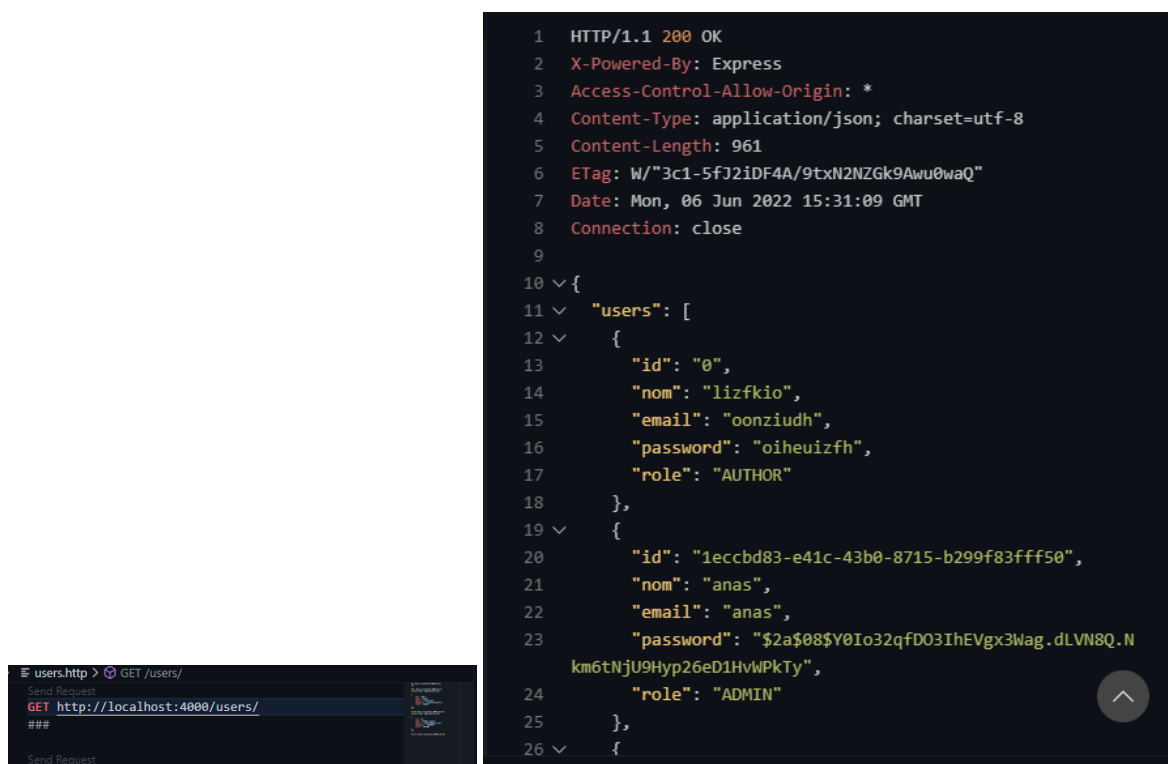
PHASE TEST

Pour tester le fonctionnement de mon projet, j'ai essayé d'installer REST Client dans visual studio code, qui va me donner une possibilité de tester les requêtes http, (**Get,Post,Delete,Patch**).

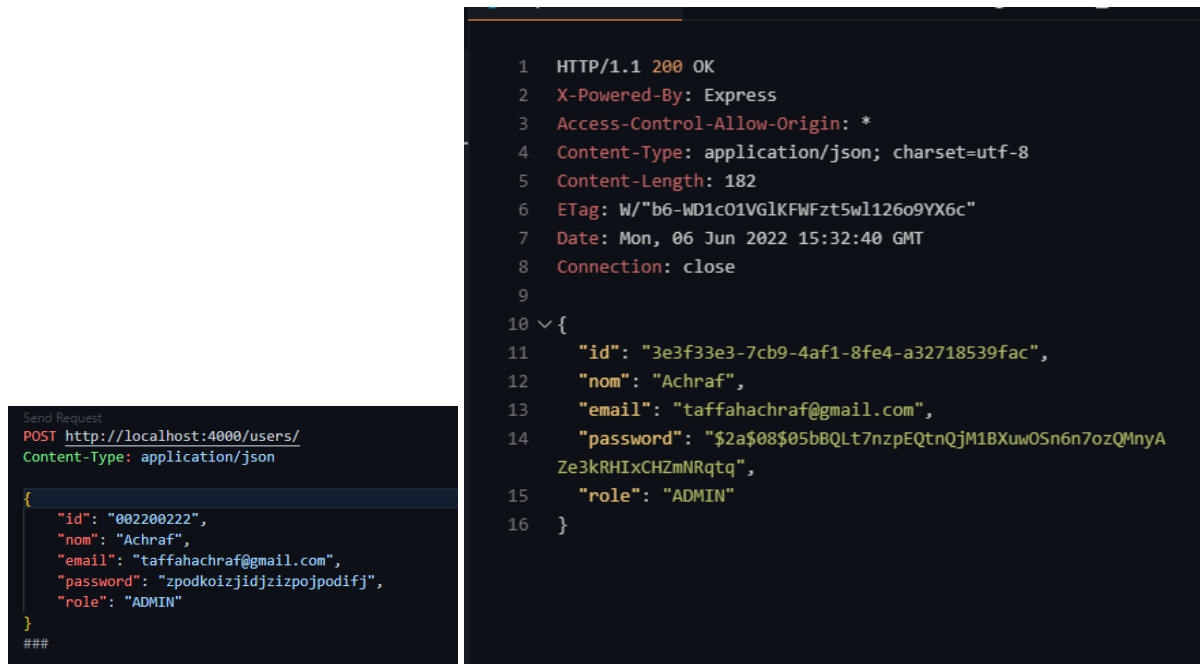
Donc j'ai créé les fichiers suivants pour tester avec tous les requêtes.



La commande **Get** donne le résultat suivant.



La commande **Post** donne le résultat suivant.



The screenshot shows a REST client interface with two panels. The left panel displays the request details: a POST to `http://localhost:4000/users/` with a JSON body containing user information. The right panel shows the response, which is a 200 OK status with a JSON body containing the created user's details.

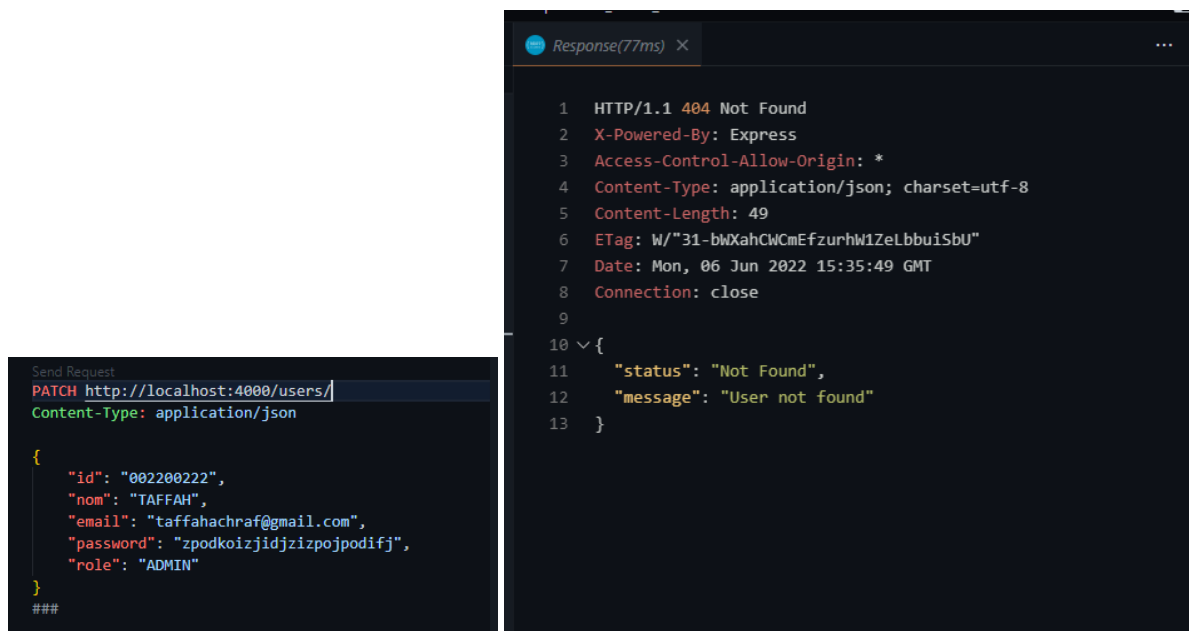
```
Send Request
POST http://localhost:4000/users/
Content-Type: application/json

{
  "id": "002200222",
  "nom": "Achraf",
  "email": "taffahachraf@gmail.com",
  "password": "zpodkoizjidjizpojpodifj",
  "role": "ADMIN"
}
###

1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Access-Control-Allow-Origin: *
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 182
6 ETag: W/"b6-WD1c01VG1KFWFzt5wL126o9YX6c"
7 Date: Mon, 06 Jun 2022 15:32:40 GMT
8 Connection: close
9
10 {
11   "id": "3e3f33e3-7cb9-4af1-8fe4-a32718539fac",
12   "nom": "Achraf",
13   "email": "taffahachraf@gmail.com",
14   "password": "$2a$08$05bBQLt7nzpEQtnQjM1BXuwOSn6n7ozQMnyAZe3kRHixCHZmNRqtq",
15   "role": "ADMIN"
16 }
```

La commande **PATCH** donne le résultat suivant :

Voilà l'erreur qui va afficher lorsque je veux afficher les informations d'un utilisateur qui n'existe pas.



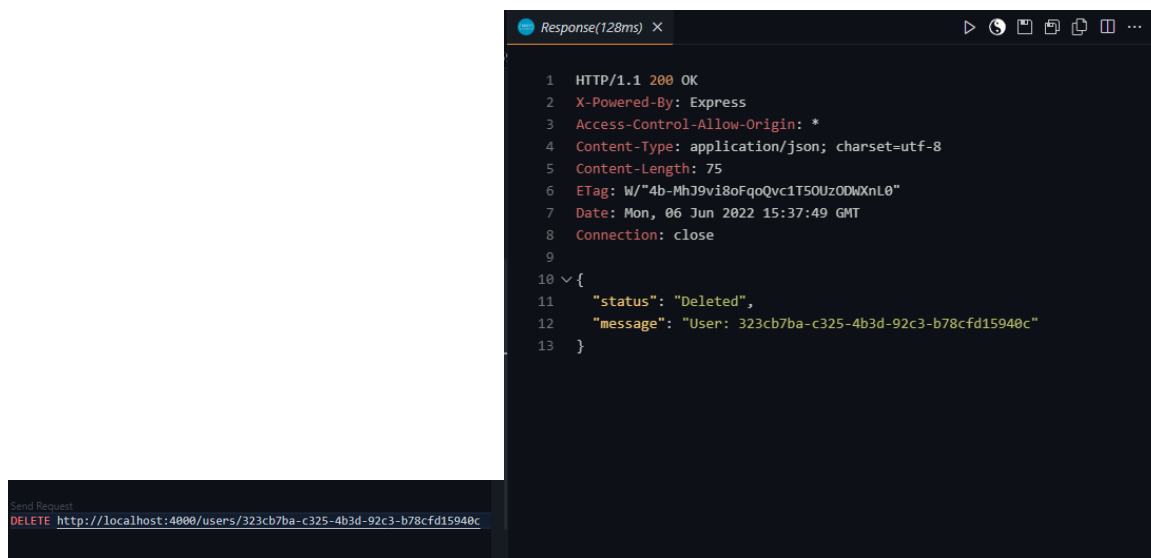
The screenshot shows a REST client interface with two panels. The left panel displays the request details: a PATCH to `http://localhost:4000/users/` with a JSON body containing user information. The right panel shows the response, which is a 404 Not Found status with a JSON body indicating the user was not found.

```
Send Request
PATCH http://localhost:4000/users/
Content-Type: application/json

{
  "id": "002200222",
  "nom": "TAFFAH",
  "email": "taffahachraf@gmail.com",
  "password": "zpodkoizjidjizpojpodifj",
  "role": "ADMIN"
}
###

Response(77ms) X
1 HTTP/1.1 404 Not Found
2 X-Powered-By: Express
3 Access-Control-Allow-Origin: *
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 49
6 ETag: W/"31-bkXahCWcEfzurhW1ZeLbbuiSbU"
7 Date: Mon, 06 Jun 2022 15:35:49 GMT
8 Connection: close
9
10 {
11   "status": "Not Found",
12   "message": "User not found"
13 }
```


La commande **DELETE** pour supprimer les informations d'un utilisateur qui donne le résultat suivant.



The screenshot shows a REST client interface with a 'Send Request' button and a 'Response(128ms)' tab. The request is a DELETE to `http://localhost:4000/users/323cb7ba-c325-4b3d-92c3-b78cfd15940c`. The response is an HTTP 200 OK with headers: `X-Powered-By: Express`, `Access-Control-Allow-Origin: *`, `Content-Type: application/json; charset=utf-8`, `Content-Length: 75`, `ETag: W/"4b-MhJ9vi8oFgoQvc1T5OUzODWxnL0"`, `Date: Mon, 06 Jun 2022 15:37:49 GMT`, and `Connection: close`. The JSON body is `{ "status": "Deleted", "message": "User: 323cb7ba-c325-4b3d-92c3-b78cfd15940c" }`.

```
1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Access-Control-Allow-Origin: *
4 Content-Type: application/json; charset=utf-8
5 Content-Length: 75
6 ETag: W/"4b-MhJ9vi8oFgoQvc1T5OUzODWxnL0"
7 Date: Mon, 06 Jun 2022 15:37:49 GMT
8 Connection: close
9
10 {
11   "status": "Deleted",
12   "message": "User: 323cb7ba-c325-4b3d-92c3-b78cfd15940c"
13 }
```

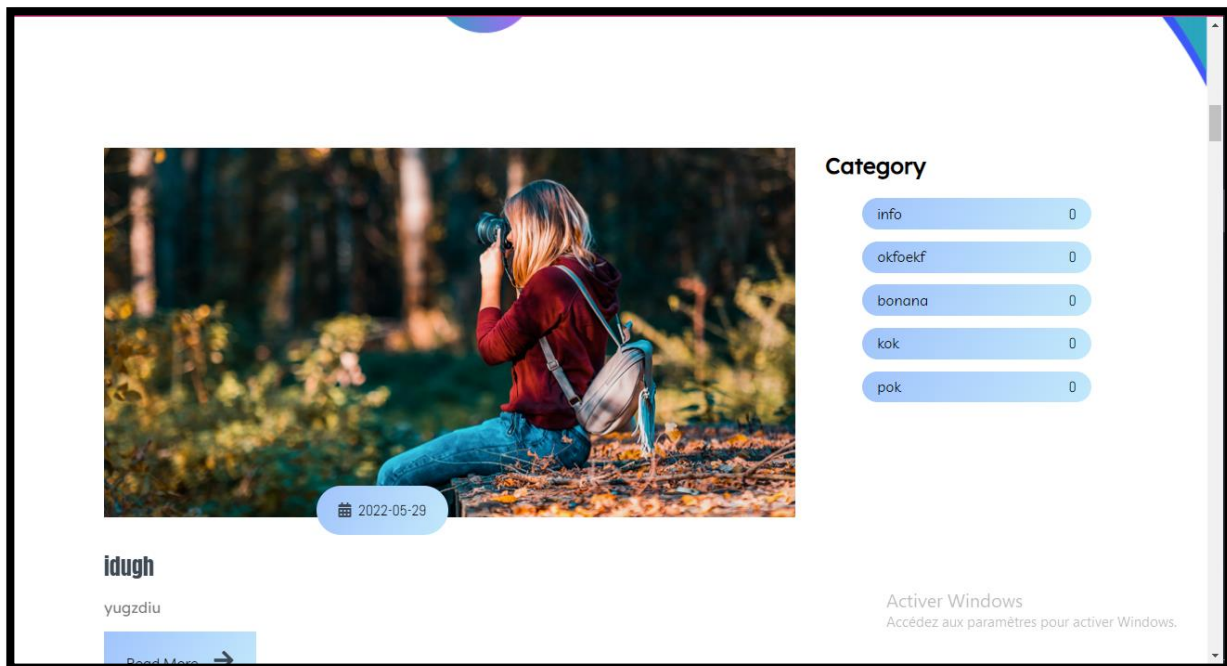
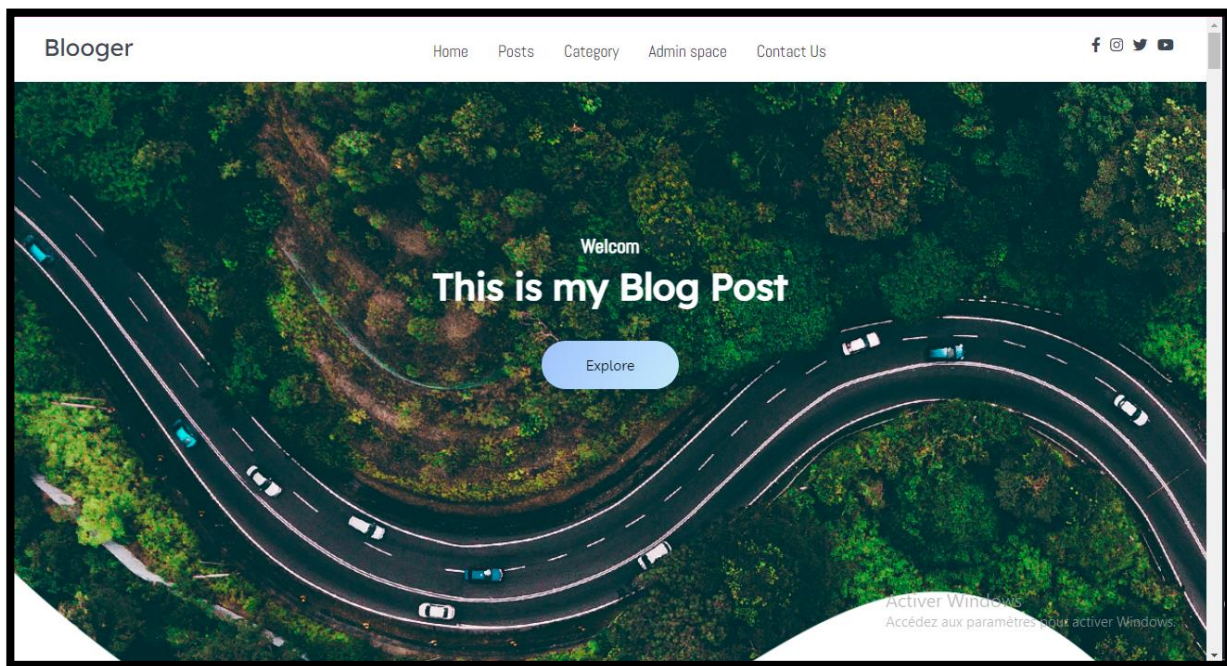
Send Request
DELETE http://localhost:4000/users/323cb7ba-c325-4b3d-92c3-b78cfd15940c

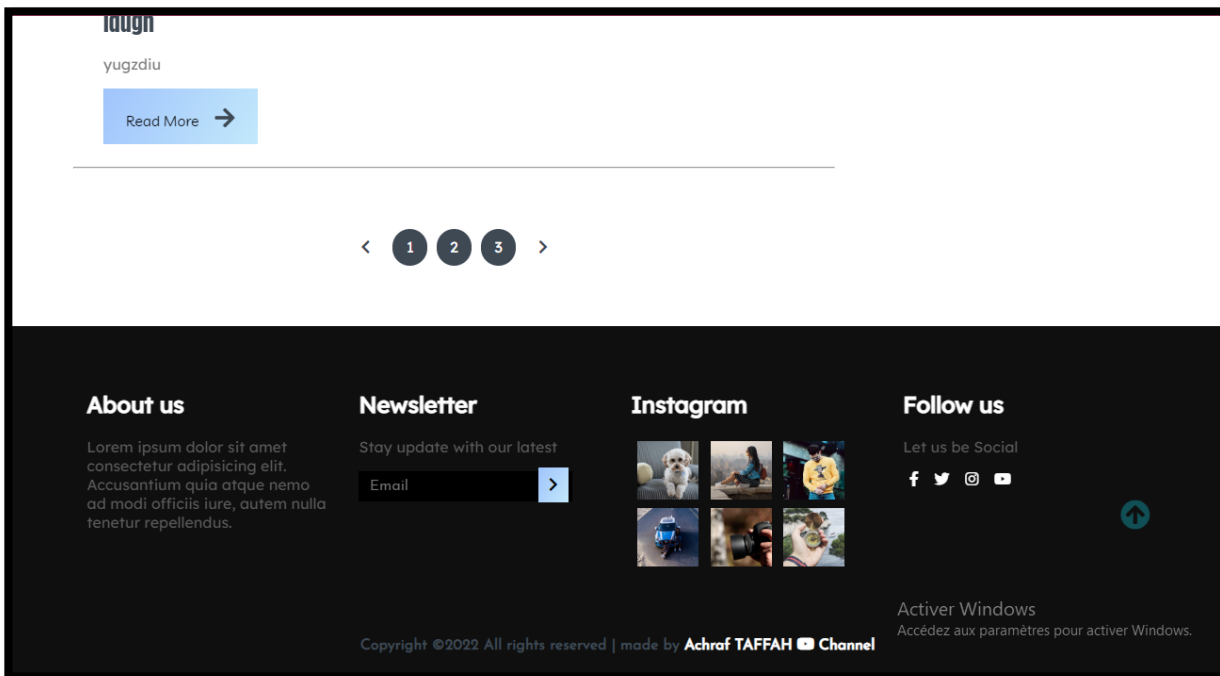
Remarque :

J'ai utilisé le même principe pour tester le fonctionnement des autres fonctionnalités, comme les commentaires, les catégories, les articles ainsi que les utilisateurs.

PHASE DEMONSTRATION

La page d'accueil





J'ai utilisé ce code pour extraire les données d'après l'API et les utilisé dans mon page web.

```
// Get all categorys
const url1="http://localhost:4000/categorie";

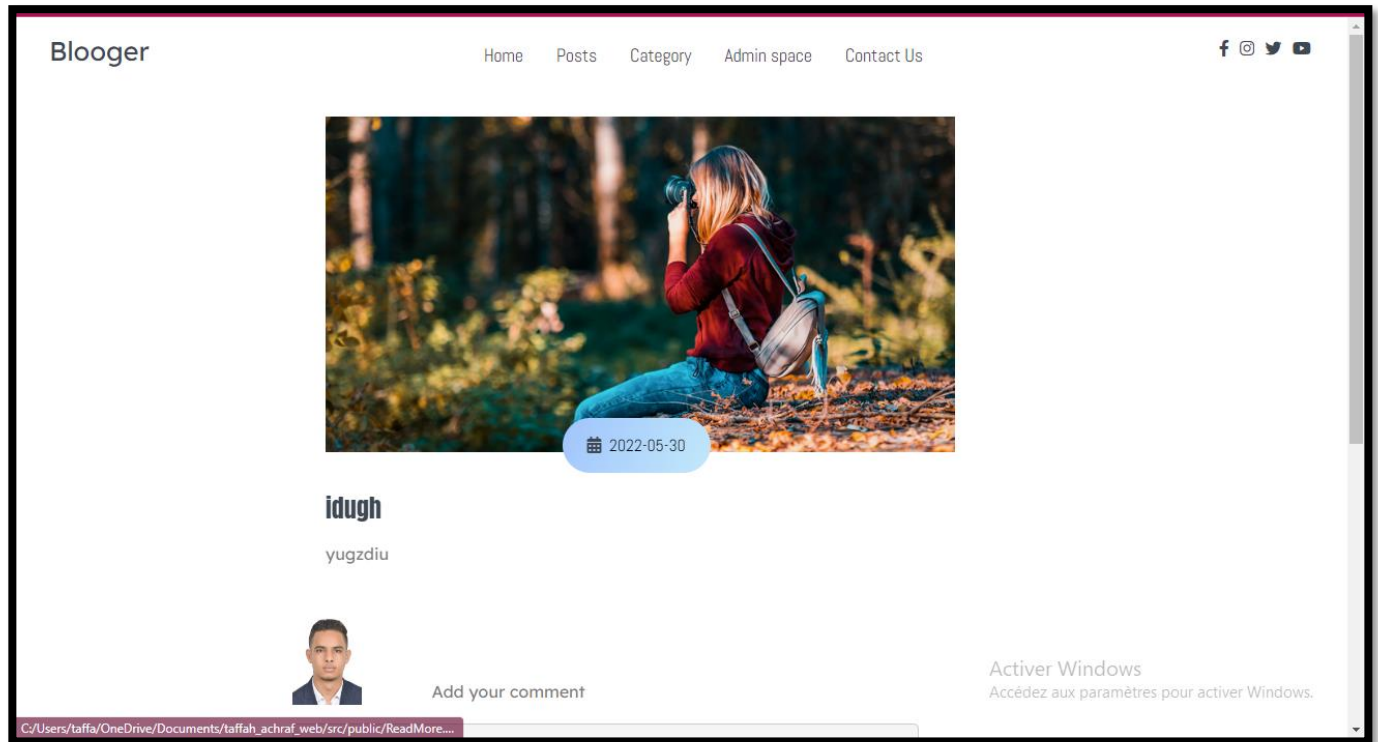
$categoriesContainer = $(".sidebar .category .category-list");
function show1(data){
  data.categories.forEach(e => {
    const dataContent = `
      <li class="list-items" data-aos="fade-left" data-aos-delay="100">
        <a href="#">${e.nom}</a>
        <span>${e._count.Articles}</span>
      </li>
    `
    $categoriesContainer.append(dataContent);
  })
}
$.getJSON(url1).then(show1);

// Get all articles
```


Alors si je click sur le Button Read More la page je consulte directement la page détail dans lequel il y a les détails de cet article.

Read More →

Donc ici l'utilisateur peut ajouter un commentaire qui concerne un article, ou bien juste lire les anciens commentaires.



Voilà le code que j'ai utilisé pour extraire les données de l'article avec un id récupère avec la méthode GET.

```
// Get id variable from url by Get methode
const queryString = window.location.search;
const urlParams = new URLSearchParams(queryString);
const i=urlParams.get('id');
if(i){
  // Get commentaires by id
  const url1="http://localhost:4000/commentaire/";
  $categoriesContainer = $(".sidebar .category .category-list");
  function show1(data2){
    data2.commentaires.forEach(e2 => {
      if(e2.articleId == i){
        const dataContent = `
          <div class="input-group">
            
            <p style="margin-top:-4%;margin-left:20%">Anasa najib</p>
            <input readonly autocomplete="off" id="main-comment" name="comment" placeholder="${e2.contenu}"
              class="comment-box"/>
          </div>
        `;
        $articlesContainer.after(dataContent);
      }
    });
  }
  $.getJSON(url1).then(show1);
}
```

```
// Get all articles
const url2 = "http://localhost:4000/article/";
$articlesContainer = $(".container .posts > hr");

function show(data) {
  const $card = $(".site-content .posts");
  data.articles.forEach(e => {
    var d = new Date(e.createdAt);
    if(e.id == i){
      console.log("hellow")
      const dataContent = `
        <div class="post-content" data-aos="zoom-in" data-aos-delay="200">
          <div class="post-image">
            <div>
              
            </div>
            <div class="post-info flex-row">
              +
              // <span><i class="fas fa-user text-gray"></i>&nbsp;&nbsp;&nbsp;${e.author.role} : ${e.author.nom}</span>
              <span><i class="fas fa-calendar-alt text-gray"></i>&nbsp;&nbsp;&nbsp;${d.toISOString().slice(0,10)}</span>
            </div>
          </div>
          <div class="post-title">
            <a href="#">${e.titre}</a>
            <p>${e.contenu}</p>
          </div>
        </div>
      `;
      $articlesContainer.before(dataContent);
    }
  });
}
$.getJSON(url2).then(show);
}
```

L'authentification

Pour vérifier le rôle de l'utilisateur à travers un login et un mot de passe et le diriger vers leur espace, j'ai utilisé le code Javascript suivant.

```
function login(){
  const email = document.getElementById('email').value;
  const password = document.getElementById('password').value;
  const url="http://localhost:4000/users/";

  function show(data) {
    data.users.forEach(e => {
      if(e.email===email && e.password===password){
        sessionStorage.setItem('u_id',e.id);
        sessionStorage.setItem('u_role',e.role);
        if(e.role === "ADMIN"){
          window.location.href="index.html";
        }
        else if(e.role === "AUTHOR"){
          window.location.href="author/index.html";
        }
        else {
          window.location.href="login.html?authentification_feiled";
        }
      }
    });
  };

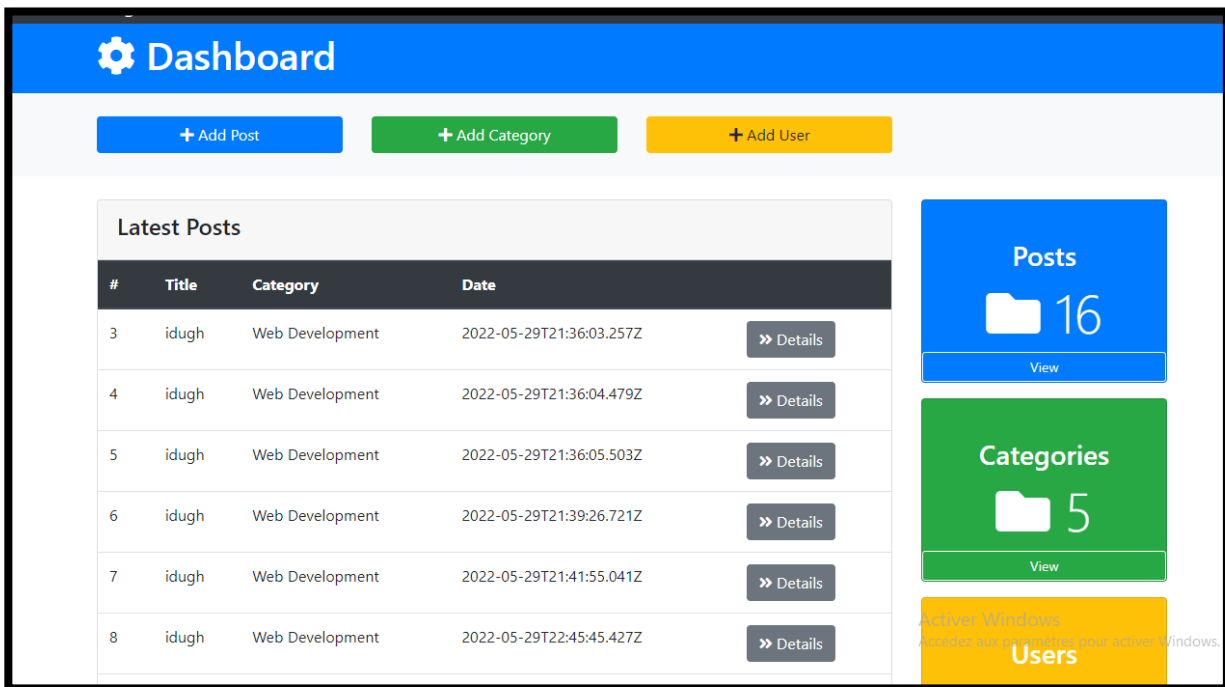
  $.getJSON(url).then(show);
}
```

Donc si par exemple j'ai saisi dans le formulaire des données qui ont existé dans la base de données comme suivant :


```
{
  "id": "3e3f33e3-7cb9-4af1-8fe4-a32718539fac",
  "nom": "Achraf",
  "email": "taffahachraf@gmail.com",
  "password": "$2a$08$05b8QLt7nzpEQtnQjM1BXuwOSn6n7ozQMnyAZe3kRHIxCHZmNRqtq",
  "role": "ADMIN"
},
```

Dans l'utilisateur va d'ériger ver l'interface de l'administrateur.

Interface de l'administrateur



Pour afficher le nombre des postes, j'ai utilisé le code javascript suivant.

```
function countArticles(){
  $articleContainer = $("#count_post");
  var i=0;
  const url="http://localhost:4000/article/";
  function show(data){
    data.articles.forEach(e => {
      i++;
    })
    const dataContent=`
    <h3>Posts</h3>
    <h4 class="display-4">
      <i class="fas fa-folder"></i> ${i}
    </h4>
    <a href="posts.html" class="btn btn-outline-light btn-sm">View</a>
  `
    $articleContainer.after(dataContent);
  }
  $.getJSON(url).then(show);
}
countArticles();
```

Pour afficher le nombre des catégories, j'ai utilisé le code javascript suivant.

```
function countCategories(){
  $categorieContainer = $("#count_cat");
  var i=0;
  const url="http://localhost:4000/categorie/";
  function show(data){
    data.categories.forEach(e => {
      i++;
    })
    const dataContent=`
      <h3>Categories</h3>
      <h4 class="display-4">
        <i class="fas fa-folder"></i> ${i}
      </h4>
      <a href="categories.html" class="btn btn-outline-light btn-sm">View</a>
    `
    $categorieContainer.after(dataContent);
  }
  $.getJSON(url).then(show);
}
```

Pour afficher le nombre des catégories, j'ai utilisé le code javascript suivant.

```
function countUsers(){
  $userContainer = $("#count_user");
  var i=0;
  const url="http://localhost:4000/users/";
  function show(data){
    data.users.forEach(e => {
      i++;
    })
    const dataContent=`
      <h3>Users</h3>
      <h4 class="display-4">
        <i class="fas fa-folder"></i> ${i}
      </h4>
      <a href="users.html" class="btn btn-outline-light btn-sm">View</a>
    `
    $userContainer.after(dataContent);
  }
  $.getJSON(url).then(show);
}

countUsers();
```

Pour afficher la data-table avec de données qui existent dans la base de données, j'ai utilisé le code suivant.

```
function getArticles(){
  const url="http://localhost:4000/article/";
  $articlesContainer = $("#first");
  function show(data){
    data.articles.forEach(e => {
      const dataContent = `
        <tr>
          <td>${e.id}</td>
          <td>${e.titre}</td>
          <td>Web Development</td>
          <td>${e.createdAt}</td>
          <td>
            <a href="details.html?objArt=${e}" class="btn btn-secondary">
              <i class="fas fa-angle-double-right"></i> Details
            </a>
          </td>
        </tr>
      `;
      $articlesContainer.before(dataContent);
    });
  }
  $.getJSON(url).then(show);
}

getArticles();
```

Code source

Vous pouvez trouver le code source de cette application sur GitHub avec le lien suivant :

<https://github.com/TAFFAHACHRAF/taffah-projet-web>

Quelques technologies utilisées

Node.js est une plateforme logicielle libre en JavaScript, orientée vers les applications réseau évènementielles hautement concurrentes qui doivent pouvoir monter en charge. Elle utilise la machine virtuelle V8, la librairie libuv pour sa boucle d'évènements, et implémente sous licence MIT les spécifications CommonJS.

En général, les **ORM** sont des bibliothèques qui font correspondre les tables de votre base de données aux classes du langage que vous utilisez pour écrire votre programme. **Prisma**, quant à lui, est une boîte à outils de base de données.

Express.js est un framework pour construire des applications web basées sur Node.js. C'est de fait le Framework standard pour le développement de serveur en Node.js.

Le but de **Bootstrap** est de permettre, de rendre facilement un site responsive design (adapté à tous les écrans : ordinateur, mobile, tablettes) sans avoir besoin de coder toute la partie CSS.

JQuery est une bibliothèque JavaScript libre et multiplateforme créée pour faciliter l'écriture de scripts côté client dans le code HTML des pages web. La première version est lancée en janvier 2006 par John Resig.

MySQL est un système de gestion de base de données relationnelle open source. Il s'agit d'un service de base de données entièrement géré pour déployer des applications natives du cloud.

PhpMyAdmin est un outil d'administration gratuit et open source pour MySQL.

CONCLUSION

Ce projet ma permet d'exploiter et de mettre en avant mes atouts et mes connaissances acquis, de découvrir une approche en matière de conception et d'analyse, de respecter des délais pour les documents à rendre et d'avoir une nouvelle expérience dans le monde informatique et d'approfondir mes connaissances en développement web, ainsi de découvrir des nouvelles technologies qui je ne connaissais pas avant pour la résolution des certains problèmes. Mon projet a bien respecté les demandes du cahier des charges. J'ai prêté beaucoup d'attention à la qualité et la lisibilité du code source afin de créer un site web performant et évolutive. Enfin, le projet reste une initialisation pour future développements. En perspectives cette application pourrait être améliorée et enrichie par des fonctionnalités avancées.

RÉFÉRENCES & BIBLIOGRAPHIE

<https://jquery.daaif.net/>

<https://www.prisma.io/docs/getting-started/setup->

<https://www.phpmyadmin.net/>

<https://ampps.com/>

<https://expressjs.com/fr/>

ENSET MOHAMMEDIA

Projet Web

Année universitaire 2021-2022