

TP12 – Express+Prisma

| | |
|---|----------|
| 1. TOOLS | 2 |
| 1.1. Node modules list | 2 |
| 1.2. Mise en place | 2 |
| 1.2.1. Commandes de depart | 2 |
| 1.2.2. Scripts | 2 |
| 1.2.3. Nettoyage de départ..... | 2 |
| 2. LECTURE DES DONNÉES..... | 2 |
| 2.1. Créer une interface de base | 2 |
| 2.2. Données Arduino | 3 |
| 2.2.1. Données envoyées..... | 3 |
| 2.2.2. Récupérer les données | 3 |
| 2.2.3. Afficher sur la page web | 3 |
| 2.3. Fichier de refresh AJAX..... | 4 |
| 2.4. Ecrire dans la db | 4 |

1. Tools

1.1. Node modules list

nodemon
express
express-generator
serialport
prisma

1.2. Mise en place

1.2.1. Commandes de depart

Créer et aller dans un dossier, puis :

```
express --view=twig .  
npm i  
npm i serialport prisma
```

1.2.2. Scripts

Ajouter dans le `package.json` :

```
"scripts": {  
  "start": "node ./bin/www",  
  "dev": "nodemon -e js,json,twig"  
},
```

```
npm run dev
```

1.2.3. Nettoyage de départ

Supprimer les fichiers et variables inutiles :

- `/public/stylesheets/styles.css`
- `/routes/users.js`
- Dans le fichier `app.js` :

```
var usersRouter = require('./routes/users');  
app.use('/users', usersRouter);
```

2. Lecture des données

2.1. Créer une interface de base

Dans `/views/index.twig` :

```
{% extends 'layout.twig' %}  
  
{% block body %}  
<h1>Valeur lue</h1>  
<p>Mesure : <span id="mesure-lue"> {{ mesure }}</span></p>  
{% endblock %}
```

2.2. Données Arduino

2.2.1. Données envoyées

Injecter un code simple dans un Arduino, avec un potentiomètre sur A0 :

```
#define pot A0
int mesure;
void setup() {
  Serial.begin(9600);
}

void loop() {
  delay(1000);
  mesure = analogRead(pot);
  Serial.println(mesure, DEC);
}
```

2.2.2. Récupérer les données

Voir : <https://serialport.io/docs/guide-usage>

Dans le fichier /routes/index.js :

```
const { SerialPort } = require("serialport");
const { ReadlineParser } = require("@serialport/parser-readline");

// Ouverture d'une communication avec le port COM3 à 9600 bauds
const port = new SerialPort({
  path: "COM3",
  baudRate: 9600,
});

// La lecture série se fera jusqu'à rencontrer un retour à la ligne
const parser = port.pipe(new ReadlineParser({ delimiter: "\n" }));

let lastMeasure;

parser.on("data", data => {
  console.log(`Data: ${data}`);
  lastMeasure = data;
});
```

À ce stade, on récupère une valeur toutes les secondes dans la console, côté serveur.

2.2.3. Afficher sur la page web

Dans le fichier /routes/index.js, changer le `res.render` de façon à passer la variable `lastMeasure` à l'index :

```
router.get("/", function (req, res, next) {
  res.render("index", { mesure: lastMeasure });
});
```

Malheureusement, la page ne se met pas à jour toute seule. Passer `lastMeasure` de cette façon ne sert pas à grand-chose.

```
res.render("index", { mesure: lastMeasure });
```

2.3. Fichier de refresh AJAX

Pour avoir une mise à jour sans avoir à recharger la page, on utilisera la fonction `jQuery.ajax()` :

```
// setInterval() se lance toutes les xxx millisecondes
setInterval(() => {
  $.ajax({
    type: "post",
    url: "/api/mesure",
    dataType: "json",
    success: function (response) {
      $("#measure-lue").text(response);
    },
  });
}, 1000);
```

L'url `"/api/mesure"` n'existe pas encore, créons-le dans `/routes/index.js` :

```
router.post("/api/mesure", (req, res) => {
  res.json(lastMeasure);
});
```

2.4. Ecrire dans la db

```
npx prisma init
```

Ajouter/changer dans `prisma.schema` :

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

model Measure {
  id      Int @id @default(autoincrement())
  measure Int
}
```

On génère ensuite les modèles dans les fichiers qui créeront la db :

```
npx prisma generate
```

Après la génération, on crée le fichier de db et les tableaux doivent s'y trouver :

```
npx prisma db push
```

Changer la fonction dans pour aussi insérer la mesure dans la db :

```
const { PrismaClient } = require("@prisma/client");
const prisma = new PrismaClient();

parser.on("data", async data => {
  console.log(`Data: ${data}`);
  lastMeasure = data;
  const measure = await prisma.measure.create({
    data: {
```

```
        mesure: Number(lastMesure),  
      },  
    });  
    console.log(mesure);  
  });
```

Maintenant, toutes les mesures seront inscrites dans la db !