

<b>1. Chapitre 1: Etude de projet.....</b>	<b>3</b>
1.1. Présentation de l'Application.....	3
1.2. Objectif.....	3
1.3. Besoins de marché.....	3
1.4. Public Cible.....	4
2. Présentation de projet.....	4
2.1. Capture de besoin.....	4
2.1.1. Spécifications des Besoins Fonctionnels.....	4
2.1.2. Spécifications des Besoins non Fonctionnels.....	5
2.2. Environnement de travail.....	6
2.2.1. environnement logiciel.....	6
2.2.2. environnement matériel.....	7
2.3. Architecture.....	7
2.3.1. Architecture de système.....	7
Couche de présentation:.....	7
Couche métier:.....	8
Couche d'accès aux données:.....	8
2.3.2. architecture de l'application.....	8
<b>Conclusion.....</b>	<b>10</b>
<b>Chapitre 2: Partie Front-end.....</b>	<b>11</b>
1) Configurer le projet Angular 16.....	11
2) Configurer le module d'application.....	12
3) Définir des itinéraires pour Angular AppRoutingModuleModule.....	13
4) Définir Model Class : Model tutoriel contient 4 attributs.....	13
5) Créer un service de données.....	14
Technology.....	14
<b>6) Créer des composants angulaires 16.....</b>	<b>16</b>
a) Importer Bootstrap dans le projet Angular 16.....	16
b) Ajouter une barre de navigation et une vue de routeur à l'exemple Angular 16 CRUD... 16	
<b>c) Ajouter un nouveau tutoriel.....</b>	<b>18</b>
<b>d) Liste des tutoriels.....</b>	<b>22</b>
Exécutez l'application Angular 16.....	31
<b>conclusion.....</b>	<b>31</b>
1) Créer une application Node.js.....	31
2) Configurer le serveur Web Express.....	33
3) Configurer la base de données PostgreSQL et Sequelize.....	34
4) Initialiser Sequelize.....	35
5) Définir le modèle Sequelize.....	37
6) Créer le contrôleur.....	38
7) Définir des itinéraires (routes).....	38
8) Testez les API.....	40
a) Créer un nouveau didacticiel à l'aide de l'API POST/tutorials.....	41

b) Récupérez tous les didacticiels à l'aide de l'API GET /tutorials.....	42
c) Récupérez un seul tutoriel par identifiant en utilisant GET /tutorials/:id Api.....	42
d) Mettre à jour un tutoriel à l'aide de PUT /tutorials/:id =2.....	43
e) Recherchez tous les didacticiels publiés à l'aide de GET /tutorials/published Api.....	43
f) Supprimer un didacticiel à l'aide de DELETE /tutorials/:id Api.....	44
g) Supprimez tous les didacticiels à l'aide de l'API DELETE /tutorials.....	45
<b>9) Conclusion.....</b>	<b>45</b>
<b>Chapitre 3 : Partie Back-end :.....</b>	<b>46</b>
1) Créer une application Node.js.....	46
2) Configurer le serveur Web Express.....	47
3) Configurer la base de données PostgreSQL et Sequelize.....	49
4) Initialiser Sequelize.....	50
5) Définir le modèle Sequelize.....	52
6) Créer le contrôleur.....	53
7) Définir des itinéraires (routes).....	57
8) Testez les API.....	59
a) Créer un nouveau didacticiel à l'aide de l'API POST/tutorials.....	60
b) Récupérez tous les didacticiels à l'aide de l'API GET /tutorials.....	61
c) Récupérez un seul tutoriel par identifiant en utilisant GET /tutorials/:id Api.....	61
d) Mettre à jour un tutoriel à l'aide de PUT /tutorials/:id =2.....	62
e) Recherchez tous les didacticiels publiés à l'aide de GET /tutorials/published Api.....	62
f) Supprimer un didacticiel à l'aide de DELETE /tutorials/:id Api.....	63
g) Supprimez tous les didacticiels à l'aide de l'API DELETE /tutorials.....	64
<b>9) Conclusion.....</b>	<b>64</b>

# Application de publication des tutoriels informatiques

## 1. Chapitre 1: Etude de projet

### 1.1. Présentation de l'Application

Notre objectif est de créer une application conviviale qui permettra aux utilisateurs de publier, rechercher et consulter une multitude de tutoriels sur différents sujets. Que ce soit dans le but d'acquérir de nouvelles compétences, de résoudre des problèmes techniques ou simplement de découvrir de nouvelles passions, notre application sera le lieu parfait où les utilisateurs pourront découvrir des guides détaillés et des instructions pratiques.

### 1.2. Objectif

Cette application vise principalement à établir une communauté dynamique et engagée axée sur le partage de connaissances et de compétences. L'objectif de notre entreprise est de proposer un cadre propice à la connexion, à l'interaction et à l'apprentissage mutuel grâce à des tutoriels de grande qualité.

### 1.3. Besoins de marché

Lorsque l'apprentissage en ligne et la consommation de contenu numérique s'intensifient, il y a une demande grandissante pour des plateformes qui permettent d'accéder facilement à des tutoriels bien organisés et fiables. L'objectif de notre application est de satisfaire cette demande en offrant une interface conviviale et des fonctionnalités solides qui rendent l'acquisition et le partage de connaissances accessibles à tous.

## 1.4. Public Cible

Nous attirons un large éventail d'utilisateurs, allant des étudiants qui souhaitent approfondir leurs connaissances à des professionnels qui souhaitent rester informés des dernières tendances et techniques dans leur domaine respectif. Notre objectif est de concevoir une expérience utilisateur à la fois séduisante pour les novices et enrichissante pour les experts.

## 2. Présentation de projet

### 2.1. Capture de besoin

#### 2.1.1. Spécifications des Besoins Fonctionnels

##### ❖ Publication de tutoriels:

- **Création de compte utilisateur:** Les utilisateurs doivent pouvoir créer un compte et fournir des informations de base (username, email,password.).
- **Ajout de tutoriels:** Les utilisateurs doivent pouvoir ajouter des tutoriels en suivant un processus simple et intuitif.
- **Gestion des tutoriels:** Les utilisateurs doivent pouvoir modifier, supprimer et organiser leurs tutoriels.

##### ❖ Recherche et consultation de tutoriels:

- **Recherche par mot-clé:** Les utilisateurs doivent pouvoir rechercher des tutoriels par mots-clés pertinents.
- **Affichage des résultats de recherche:** Les résultats de recherche doivent être pertinents et organisés de manière claire et concise.
- **Consultation des tutoriels:** Les utilisateurs doivent pouvoir consulter les tutoriels en détail.

### 2.1.2. Spécifications des Besoins non Fonctionnels

#### ❖ **Performance:**

- L'application doit être réactive et fluide, même avec un grand nombre d'utilisateurs et de tutoriels.
- Le temps de chargement des pages et des tutoriels doit être minimal.

#### ❖ **Sécurité:**

- Les données des utilisateurs doivent être protégées contre les accès non autorisés, la modification et la suppression.
- Les transactions financières doivent être sécurisées.

#### ❖ **Disponibilité:**

- L'application doit être accessible 24h/24 et 7j/7.
- Des mesures de redondance doivent être mises en place pour garantir la disponibilité de l'application en cas de panne.

#### ❖ **Scalabilité:**

- L'application doit pouvoir supporter une croissance importante du nombre d'utilisateurs et de tutoriels.
- L'infrastructure doit pouvoir être facilement scalable pour répondre aux besoins croissants.

#### ❖ **Maintenance:**

- L'application doit être facile à maintenir et à mettre à jour.
- Une documentation complète doit être disponible pour les développeurs.

❖ **Compatibilité:**

- doit être compatible avec les différents navigateurs web.

❖ **Accessibilité:**

- doit être accessible aux personnes handicapées.

❖ **Confidentialité:**

- Les données des utilisateurs doivent être collectées et utilisées de manière conforme à la réglementation en vigueur.
- Les utilisateurs doivent avoir le contrôle de leurs données personnelles.

## 2.2. Environnement de travail

### 2.2.1. environnement logiciel

❖ **Serveur (Back-end):**

- Système d'exploitation: windows 10
- Node.js v20.0.1
- PostgreSQL v15

❖ **Client (Front-end):**

- Angular v16
- TypeScript
- HTML5
- CSS3

#### ❖ Outils:

- Git (gestion de versions)
- Postman (test API)
- Visual Studio Code (VS Code)

#### 2.2.2. environnement matériel

- ❖ Ordinateur portable ou fixe avec une configuration suffisante pour exécuter les outils de développement et la plateforme web.
- ❖ Processeur: Intel core i7 9th Gen
- ❖ Mémoire vive: 24 Go
- ❖ Espace disque: 500 GB
- ❖ Écran: 15.6" FHD (1920 x 1080), 60 Hz

### 2.3. Architecture

#### 2.3.1. Architecture de système

L'architecture du système est décomposée en trois couches principales :

##### **Couche de présentation:**

- Technologie: HTML, CSS, JavaScript, framework front-end (Angular).
- Rôle: Affichage de l'interface tutoriels et interaction avec l'utilisateur.
- Fonctionnalités:
  - Formulaire de saisie et de modification des tutoriels.
  - Affichage des tutoriels.
  - Gestion des interactions utilisateur (clics, navigation, etc.).

##### **Couche métier:**

- Technologie: javascript, node js, framework d'API (RESTful).
- Rôle: traitement des requêtes, logique métier et accès aux données.
- Fonctionnalités:
  - Authentification et sécurité des utilisateurs.

- Gestion des comptes utilisateurs et des tutoriels.
- Consultation et affichage des tutoriels.
- Communication avec la couche d'accès aux données.

### **Couche d'accès aux données:**

- Technologie: PostgreSQL
- Rôle: Stockage et récupération des données.
- **Fonctionnalités:**
  - Stockage des informations des utilisateurs et tutoriels.
  - Gestion des requêtes SQL.
  - Optimisation des performances des accès aux données.

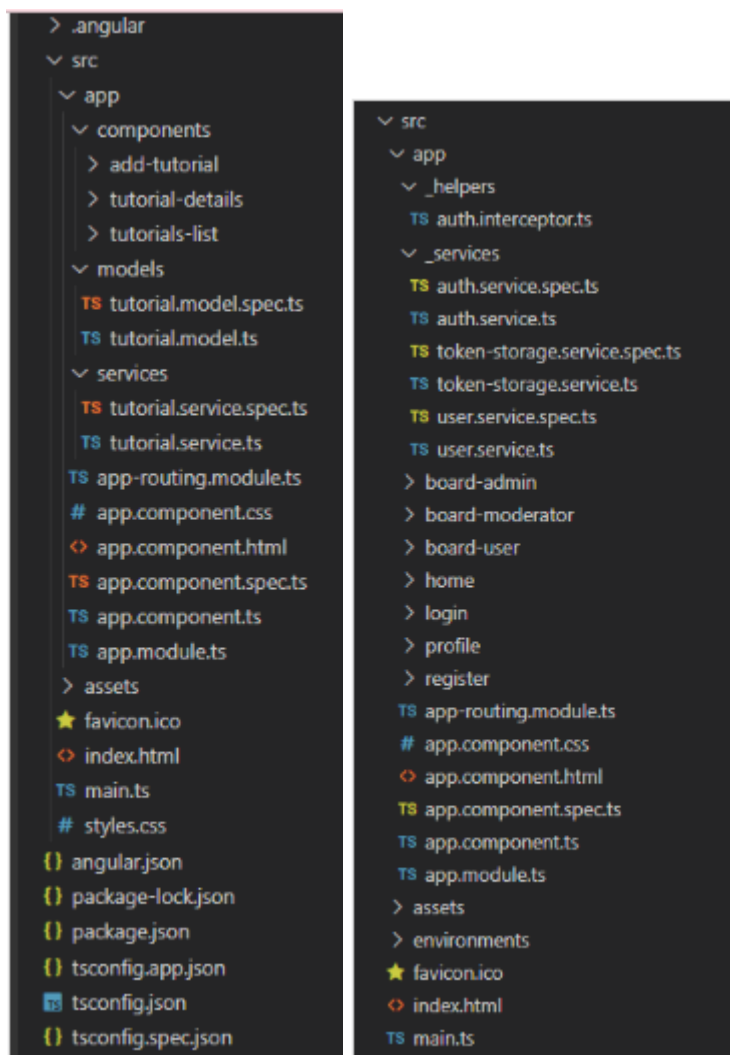
### 2.3.2. architecture de l'application

L'architecture de l'application peut être conçue selon un pattern MVC (Model-View-Controller) :

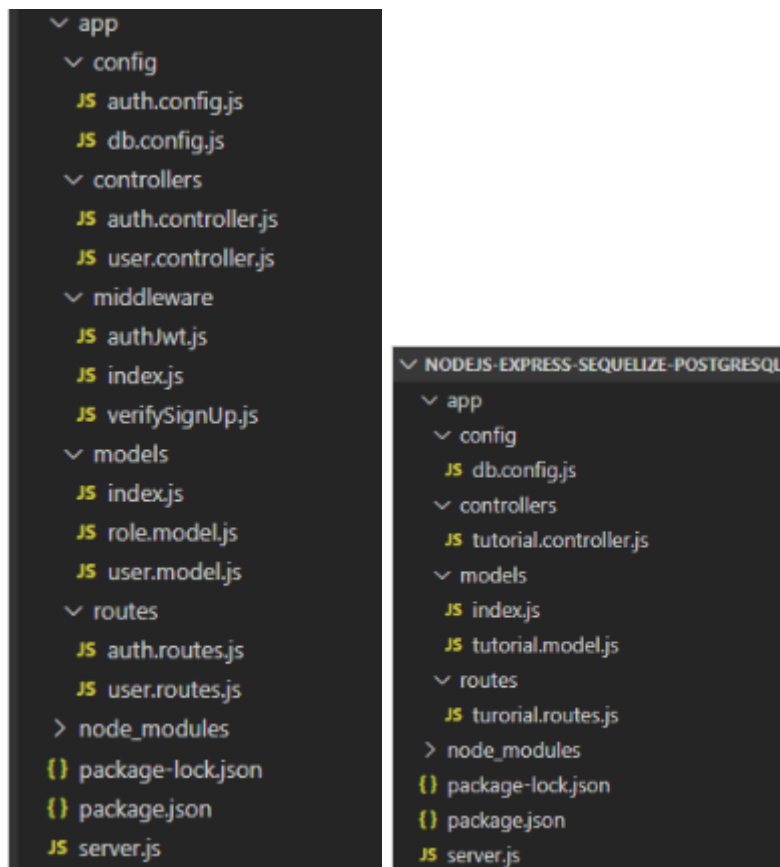
- **Modèle** : Représentation des données et de la logique métier.
- **Vue** : Interface utilisateur et interactions avec l'utilisateur.
- **Contrôleur** : Intermédiaire entre la vue et le modèle, gérant les requêtes et les actions.



→ Architecture de frontend :



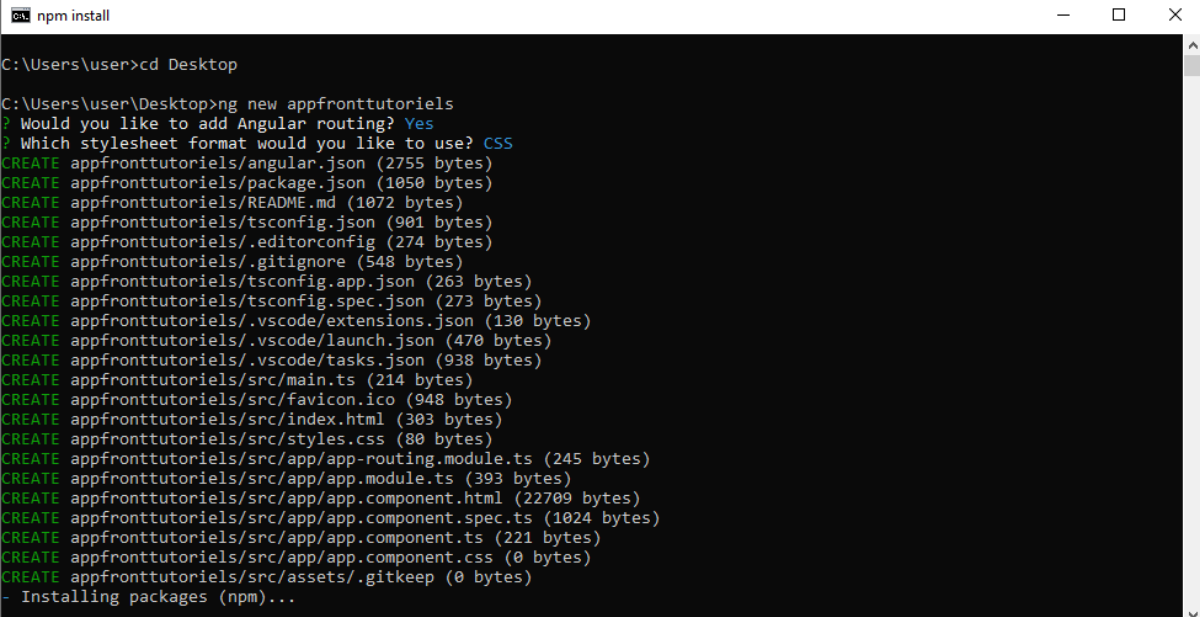
→ Architecture de backend



## Conclusion

Ce chapitre présente les bases du projet, ses objectifs, ses besoins et son environnement de travail. Les chapitres suivants approfondiront l'architecture du frontend, la conception de la base de données et l'implémentation des fonctionnalités de l'application.

## Chapitre 2: Partie Front-end



```
npm install

C:\Users\user>cd Desktop

C:\Users\user\Desktop>ng new appfronttutoriels
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE appfronttutoriels/angular.json (2755 bytes)
CREATE appfronttutoriels/package.json (1050 bytes)
CREATE appfronttutoriels/README.md (1072 bytes)
CREATE appfronttutoriels/tsconfig.json (901 bytes)
CREATE appfronttutoriels/.editorconfig (274 bytes)
CREATE appfronttutoriels/.gitignore (548 bytes)
CREATE appfronttutoriels/tsconfig.app.json (263 bytes)
CREATE appfronttutoriels/tsconfig.spec.json (273 bytes)
CREATE appfronttutoriels/.vscode/extensions.json (130 bytes)
CREATE appfronttutoriels/.vscode/launch.json (470 bytes)
CREATE appfronttutoriels/.vscode/tasks.json (938 bytes)
CREATE appfronttutoriels/src/main.ts (214 bytes)
CREATE appfronttutoriels/src/favicon.ico (948 bytes)
CREATE appfronttutoriels/src/index.html (303 bytes)
CREATE appfronttutoriels/src/styles.css (80 bytes)
CREATE appfronttutoriels/src/app/app-routing.module.ts (245 bytes)
CREATE appfronttutoriels/src/app/app.module.ts (393 bytes)
CREATE appfronttutoriels/src/app/app.component.html (22709 bytes)
CREATE appfronttutoriels/src/app/app.component.spec.ts (1024 bytes)
CREATE appfronttutoriels/src/app/app.component.ts (221 bytes)
CREATE appfronttutoriels/src/app/app.component.css (0 bytes)
CREATE appfronttutoriels/src/assets/.gitkeep (0 bytes)
- Installing packages (npm)...
```

### 1) Configurer le projet Angular 16

Ouvrons cmd et utilisons Angular CLI pour créer un nouveau projet Angular en tant que commande suivante :

```
ng new angular-16-crud
```

```
? Would you like to add Angular routing? Yes
```

```
? Which stylesheet format would you like to use? CSS
```

Nous devons également générer certains composants et services :

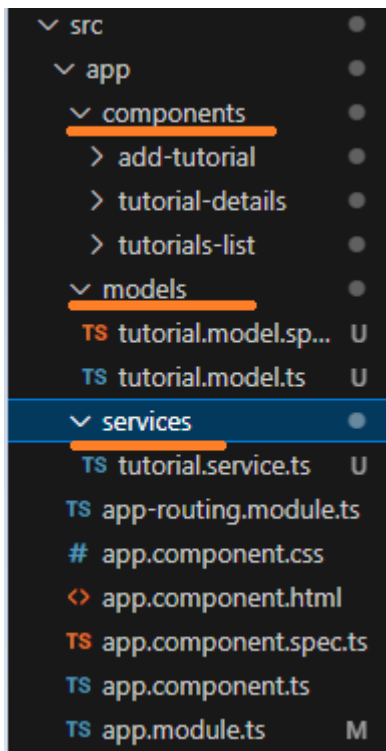
```
ng g s services/tutorial
```

```
ng g c components/add-tutorial
```

```
ng g c components/tutorial-details
```

```
ng g c components/tutorials-list
```

```
ng g class models/tutorial --type=model
```



## 2) Configurer le module d'application

Open *app.module.ts* and import `FormsModule`, `HttpClientModule`

```
TS app.module.ts M X
src > app > TS app.module.ts > ...
5 import { AppComponent } from './app.component';
6 import { AddTutorialComponent } from './components/add-tutorial/add-tutorial.component';
7 import { TutorialDetailsComponent } from './components/tutorial-details/tutorial-details.component';
8 import { TutorialsListComponent } from './components/tutorials-list/tutorials-list.component';
9 import { FormsModule } from '@angular/forms';
10 import { HttpClientModule } from '@angular/common/http';
11 @NgModule({
12   declarations: [
13     AppComponent,
14     AddTutorialComponent,
15     TutorialDetailsComponent,
16     TutorialsListComponent
17   ],
18   imports: [
19     BrowserModule,
20     AppRoutingModule,
21     FormsModule,
22     HttpClientModule
23   ],
24   providers: [],
25   bootstrap: [AppComponent]
26 })
27 export class AppModule { }
```

### 3) Définir des itinéraires pour Angular AppRoutingModule

Il existe 3 itinéraires (path) principaux :

- /tutorials for tutorials-list component
- /tutorials/:id for tutorial-details component
- /add for add-tutorial component

app-routing.module.ts

```
TS app-routing.module.ts M X
src > app > TS app-routing.module.ts > ...
1 import { NgModule } from '@angular/core';
2 import { RouterModule, Routes } from '@angular/router';
3 import { TutorialsListComponent } from '../components/tutorials-list/tutorials-list.component';
4 import { TutorialDetailsComponent } from '../components/tutorial-details/tutorial-details.component';
5 import { AddTutorialComponent } from '../components/add-tutorial/add-tutorial.component';
6
7 const routes: Routes = [
8   { path: '', redirectTo: 'tutorials', pathMatch: 'full' },
9   { path: 'tutorials', component: TutorialsListComponent },
10  { path: 'tutorials/:id', component: TutorialDetailsComponent },
11  { path: 'add', component: AddTutorialComponent }
12 ];
13
14 @NgModule({
15   imports: [RouterModule.forRoot(routes)],
16   exports: [RouterModule]
17 })
18 export class AppRoutingModule { }
19
```

### 4) Définir Model Class : Model tutoriel contient 4 attributs

- id
- title
- description
- published

```
TS tutorial.model.ts U X
src > app > models > TS tutorial.model.ts > ...
1  export class Tutorial {
2      id?: any;
3      title?: string;
4      description?: string;
5      published?: boolean;
6  }
```

## 5) Créer un service de données

Ce service utilisera Angular HttpClient pour envoyer des requêtes HTTP.

Vous pouvez voir que ses fonctions incluent les opérations CRUD et la méthode de recherche.

- Le composant App est un conteneur avec router-outlet. Il a une barre de navigation qui relie les chemins de path via routerLink.
- Le composant TutorialsList obtient et affiche les tutoriels.
- Le composant TutorialDetails a un formulaire pour modifier les détails du tutoriel basé sur :id.
- Le composant AddTutorial a un formulaire pour soumettre un nouveau tutoriel.
- Ces composants appellent des méthodes TutorialService qui utilisent Angular HttpClient pour effectuer des requêtes HTTP et recevoir des réponses.

### Technology

- Angular 16
- Angular HttpClient
- Angular Router
- Bootstrap 4

```

import { HttpClient } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Tutorial } from '../models/tutorial.model';
import { Observable } from 'rxjs';
const baseUrl = 'http://localhost:8080/api/tutorials';

@Injectable({
  providedIn: 'root'
})

export class TutorialService {

  constructor(private http: HttpClient) { }

  getAll(): Observable<Tutorial[]> {
    return this.http.get<Tutorial[]>(baseUrl);
  }

  get(id: any): Observable<Tutorial> {
    return this.http.get(`${baseUrl}/${id}`);
  }

  create(data: any): Observable<any> {
    return this.http.post(baseUrl, data);
  }

  update(id: any, data: any): Observable<any> {
    return this.http.put(`${baseUrl}/${id}`, data);
  }

  delete(id: any): Observable<any> {
    return this.http.delete(`${baseUrl}/${id}`);
  }

  deleteAll(): Observable<any> {
    return this.http.delete(baseUrl);
  }

  findByTitle(title: any): Observable<Tutorial[]> {
    return this.http.get<Tutorial[]>(`${baseUrl}?title=${title}`);
  }
}

```

## 6) Créer des composants angulaires 16

### a) Importer Bootstrap dans le projet Angular 16

Exécutez la commande : `npm install bootstrap@4.6.2`.

Ensuite, ouvrez `src/style.css` et ajoutez le code suivant :

```
@import "~bootstrap/dist/css/bootstrap.css";
```

### b) Ajouter une barre de navigation et une vue de routeur

Ouvrons `src/app.component.html`, ce composant App est le conteneur racine de notre application, il contiendra un élément nav.

`app.component.html`

```
<div>
  <nav class="navbar navbar-expand navbar-dark bg-success">
    <a href="#" class="navbar-brand">Amine</a>
    <div class="navbar-nav mr-auto">
      <li class="nav-item">
        <a routerLink="tutorials" class="nav-link
text-light">Tutorials</a>
      </li>
      <li class="nav-item">
        <a routerLink="add" class="nav-link text-light">Add</a>
      </li>
    </div>
  </nav>

  <div class="container mt-3">
    <router-outlet></router-outlet>
  </div>

  <!-- Informations sur les frameworks de développement -->
  <div class="framework-info">
    <h2>Frameworks de Développement</h2>

    <!-- React.js -->
```



```

<div>
  <h3>React.js</h3>
  
  <p>React.js est une bibliothèque JavaScript pour la construction
d'interfaces utilisateur.</p>
</div>

<!-- Angular -->
<div>
  <h3>Angular</h3>
  
  <p>Angular est une plateforme de développement d'applications web
développée par Google.</p>
</div>

<!-- Vue.js -->
<div>
  <h3>Vue.js</h3>
  
  <p>Vue.js est un framework JavaScript progressif pour la création
d'interfaces utilisateur.</p>
</div>

<!-- Express.js -->
<div>
  <h3>Express.js</h3>
  
  <p>Express.js est un framework web pour Node.js utilisé pour
construire des applications web et des API.</p>
</div>

<!-- Django -->
<div>
  <h3>Django</h3>
  
  <p>Django est un framework web Python open source qui encourage
le développement rapide et le design propre et pragmatique.</p>
</div>
</div>

```

## Frameworks de Développement

React.js



React.js est une bibliothèque JavaScript pour la construction d'interfaces utilisateur.

Angular



Angular est une plateforme de développement d'applications web développée par Google.

Vue.js



### c) Ajouter un nouveau tutoriel

Ce composant dispose d'un formulaire pour soumettre un nouveau tutoriel avec 2 champs : titre et description. Il appelle la méthode `TutorialService.create()`.

**components/add-tutorial/add-tutorial.component.ts**

```
import { Component, OnInit } from '@angular/core';
import { Tutorial } from 'src/app/models/tutorial.model';
import { TutorialService } from 'src/app/services/tutorial.service';

@Component({
  selector: 'app-add-tutorial',
  templateUrl: './add-tutorial.component.html',
  styleUrls: ['./add-tutorial.component.css']
})
export class AddTutorialComponent implements OnInit{
```

```

tutorial: Tutorial = {
  title: '',
  description: '',
  published: false
};
submitted = false;

constructor(private tutorialService: TutorialService) { }

ngOnInit(): void {
}

saveTutorial(): void {
  const data = {
    title: this.tutorial.title,
    description: this.tutorial.description
  };

  this.tutorialService.create(data)
    .subscribe({
      next: (res) => {
        console.log(res);
        this.submitted = true;
      },
      error: (e) => console.error(e)
    });
}

newTutorial(): void {
  this.submitted = false;
  this.tutorial = {
    title: '',
    description: '',
    published: false
  };
}
}

```

### **components/add-tutorial/add-tutorial.component.html**

```

<div>
  <div class="submit-form">

```

```

<div *ngIf="!submitted">
  <div class="form-group">
    <label for="title">Title</label>
    <input
      type="text"
      class="form-control"
      id="title"
      required
      [(ngModel)]="tutorial.title"
      name="title"
    />
  </div>

  <div class="form-group">
    <label for="description">Description</label>
    <textarea
      class="form-control"
      id="description"
      required
      [(ngModel)]="tutorial.description"
      name="description"
      rows="4"
    ></textarea>
  </div>

  <button (click)="saveTutorial()" class="btn
btn-success">Submit</button>
</div>

<div *ngIf="submitted">
  <h4>Tutorial was submitted successfully!</h4>
  <button class="btn btn-success"
(click)="newTutorial()">Add</button>
</div>
</div>
</div>

```

Title

laravel

Description

Laravel est un framework web open-source écrit en PHP respectant le principe modèle-vue-contrôleur et entièrement développé en programmation orientée objet.

Submit

## Frameworks de Développement

Tutorial was submitted successfully!

Add

## Frameworks de Développement

Nous Trouvons le resultat dans la base de données :

The screenshot shows a PostgreSQL database interface. On the left, the 'Object Explorer' pane shows the database structure, with 'tutorials' selected under 'Tables (1)'. The main pane displays the SQL query used to create the table:

```

11 "imageUrl" character varying(255) COLLATE pg_catalog."default
12 "createdAt" timestamp with time zone NOT NULL,
13 "updatedAt" timestamp with time zone NOT NULL,
14 CONSTRAINT tutorials_pkey PRIMARY KEY (id)
15 )
16
17 TABLESPACE pg_default;
18
19 ALTER TABLE IF EXISTS public.tutorials
20 OWNER to postgres;
21 select * from tutorials;

```

Below the query, the 'Data Output' pane shows the result of the query:

id	title	description
1	laravel	Laravel est un framework web open-source écrit en PHP respectant le principe mo

## d) Liste des tutoriels

Ce composant appelle 3 méthodes TutorialService :

- getAll()
- deleteAll()
- findByTitle()

Il contient également un tutoriel de composant enfant que nous créerons plus tard.

**components/tutorials-list/tutorials-list.component.ts**

```
import { Component } from '@angular/core';
import { Tutorial } from 'src/app/models/tutorial.model';
import { TutorialService } from 'src/app/services/tutorial.service';

@Component({
  selector: 'app-tutorials-list',
  templateUrl: './tutorials-list.component.html',
  styleUrls: ['./tutorials-list.component.css']
})
export class TutorialsListComponent {
  tutorials?: Tutorial[];
  currentTutorial: Tutorial = {};
  currentIndex = -1;
  title = '';

  constructor(private tutorialService: TutorialService) { }

  ngOnInit(): void {
    this.retrieveTutorials();
  }

  retrieveTutorials(): void {
    this.tutorialService.getAll()
      .subscribe({
        next: (data) => {
          this.tutorials = data;
          console.log(data);
        },
        error: (e) => console.error(e)
      });
  }
}
```

```

    });
}

refreshList(): void {
    this.retrieveTutorials();
    this.currentTutorial = {};
    this.currentIndex = -1;
}

setActiveTutorial(tutorial: Tutorial, index: number): void {
    this.currentTutorial = tutorial;
    this.currentIndex = index;
}

removeAllTutorials(): void {
    this.tutorialService.deleteAll()
        .subscribe({
            next: (res) => {
                console.log(res);
                this.refreshList();
            },
            error: (e) => console.error(e)
        });
}

searchTitle(): void {
    this.currentTutorial = {};
    this.currentIndex = -1;

    this.tutorialService.findByTitle(this.title)
        .subscribe({
            next: (data) => {
                this.tutorials = data;
                console.log(data);
            },
            error: (e) => console.error(e)
        });
}
}

```

## components/tutorials-list/tutorials-list.component.html

```
<div class="list row">
  <div class="col-md-8">
    <div class="input-group mb-3">
      <input
        type="text"
        class="form-control"
        placeholder="Search by title"
        [(ngModel)]="title"
      />
      <div class="input-group-append">
        <button
          class="btn btn-outline-secondary"
          type="button"
          (click)="searchTitle()"
        >
          Search
        </button>
      </div>
    </div>
  </div>
  <div class="col-md-6">
    <h4>Tutorials List</h4>
    <ul class="list-group">
      <li
        class="list-group-item"
        *ngFor="let tutorial of tutorials; let i = index"
        [class.active]="i == currentIndex"
        (click)="setActiveTutorial(tutorial, i)"
      >
        {{ tutorial.title }}
      </li>
    </ul>

    <button class="m-3 btn btn-sm btn-danger"
      (click)="removeAllTutorials()">
      Remove All
    </button>
  </div>
  <div class="col-md-6">
    <app-tutorial-details
      [viewModel]="true"
      [currentTutorial]="currentTutorial"
    >
  </div>
</div>
```



```

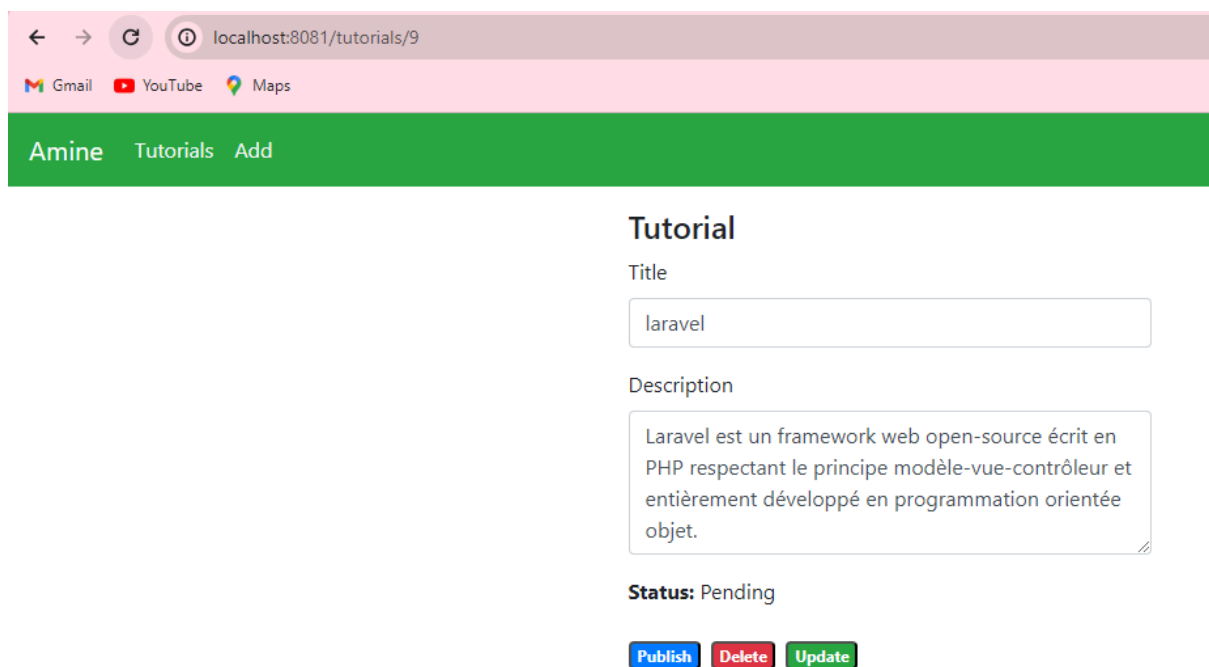
    </app-tutorial-details>
  </div>
</div>

```

Si vous cliquez sur le bouton **Edit** d'un tutoriel, vous serez dirigé vers la page du tutoriel avec l'URL : /tutorials/:id.

Voici comment nous intégrons le composant enfant :

- Nous utilisons le sélecteur de l'enfant, ici `<app-tutorial-details>`, comme directive dans le modèle parent.
- Nous utilisons la liaison de propriété pour lier les propriétés **viewMode** et **currentTutorial** de l'enfant à la propriété **currentTutorial** du parent.



← → ↻ ⓘ localhost:8081/tutorials/9

Gmail YouTube Maps

Amine Tutorials Add

### Tutorial

Title

Description

Laravel est un framework web open-source écrit en PHP respectant le principe modèle-vue-contrôleur et entièrement développé en programmation orientée objet.

**Status:** Pending

Publish
Delete
Update

Le composant détails du tutoriel ressemblera donc à ceci :

```

@Input() viewMode = false;

@Input() currentTutorial: Tutorial = {

```

```
    title: '',
    description: '',
    published: false
  };
```

Avec le décorateur **@Input()** dans la classe de composant enfant, Angular transmet la valeur de **viewMode** et **currentTutorial** à l'enfant afin que **viewMode** s'affiche comme true.

**components/tutorial-details/tutorial-details.component.ts**

```
import { Component, Input } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';
import { Tutorial } from 'src/app/models/tutorial.model';
import { TutorialService } from 'src/app/services/tutorial.service';

@Component({
  selector: 'app-tutorial-details',
  templateUrl: './tutorial-details.component.html',
  styleUrls: ['./tutorial-details.component.css']
})

export class TutorialDetailsComponent {
  @Input() viewMode = false;

  @Input() currentTutorial: Tutorial = {
    title: '',
    description: '',
    published: false
  };
  message = '';

  constructor(
    private tutorialService: TutorialService,
    private route: ActivatedRoute,
    private router: Router) { }

  ngOnInit(): void {
    if (!this.viewMode) {
      this.message = '';
      this.getTutorial(this.route.snapshot.params["id"]);
    }
  }
}
```

```

    }
}

getTutorial(id: string): void {
    this.tutorialService.get(id)
        .subscribe({
            next: (data) => {
                this.currentTutorial = data;
                console.log(data);
            },
            error: (e) => console.error(e)
        });
}

updatePublished(status: boolean): void {
    const data = {
        title: this.currentTutorial.title,
        description: this.currentTutorial.description,
        published: status
    };

    this.message = '';

    this.tutorialService.update(this.currentTutorial.id, data)
        .subscribe({
            next: (res) => {
                console.log(res);
                this.currentTutorial.published = status;
                this.message = res.message ? res.message : 'The status was
updated successfully!';
            },
            error: (e) => console.error(e)
        });
}

updateTutorial(): void {
    this.message = '';

    this.tutorialService.update(this.currentTutorial.id,
this.currentTutorial)
        .subscribe({
            next: (res) => {
                console.log(res);
            }
        });
}

```

```

        this.message = res.message ? res.message : 'This tutorial was
updated successfully!';
    },
    error: (e) => console.error(e)
  });
}

deleteTutorial(): void {
  this.tutorialService.delete(this.currentTutorial.id)
    .subscribe({
      next: (res) => {
        console.log(res);
        this.router.navigate(['/tutorials']);
      },
      error: (e) => console.error(e)
    });
}
}

```

### **components/tutorial-details/tutorial-details.component.html**

```

<div *ngIf="viewMode; else editable">
  <div *ngIf="currentTutorial.id">
    <h4>Tutorial</h4>
    <div>
      <label><strong>Title:</strong></label> {{ currentTutorial.title
    }}
  </div>
  <div>
    <label><strong>Description :</strong></label>
    {{ currentTutorial.description }}
  </div>
  <div>
    <label><strong>Status:</strong></label>
    {{ currentTutorial.published ? "Published" : "Pending" }}
  </div>
  <a
    class="badge badge-warning"
    routerLink="/tutorials/{{ currentTutorial.id }}"
  >

```

```

        Edit
      </a>
    </div>

    <div *ngIf="!currentTutorial">
      <br />
      <p>Please click on a Tutorial...</p>
    </div>
  </div>

  <ng-template #editable>
    <div *ngIf="currentTutorial.id" class="edit-form">
      <h4>Tutorial</h4>
      <form>
        <div class="form-group">
          <label for="title">Title</label>
          <input
            type="text"
            class="form-control"
            id="title"
            [(ngModel)]="currentTutorial.title"
            name="title"
          />
        </div>
        <div class="form-group">
          <label for="description">Description</label>
          <textarea
            class="form-control"
            id="description"
            [(ngModel)]="currentTutorial.description"
            name="description"
            rows="4"
          ></textarea>
        </div>

        <div class="form-group">
          <label><strong>Status:</strong></label>
          {{ currentTutorial.published ? "Published" : "Pending" }}
        </div>
      </form>

      <button
        class="badge badge-primary mr-2"

```

```

        *ngIf="currentTutorial.published"
        (click)="updatePublished(false) "
    >
        UnPublish
    </button>
    <button
        *ngIf="!currentTutorial.published"
        class="badge badge-primary mr-2"
        (click)="updatePublished(true) "
    >
        Publish
    </button>

    <button class="badge badge-danger mr-2"
(click)="deleteTutorial() ">
        Delete
    </button>

    <button
        type="submit"
        class="badge badge-success mb-2"
        (click)="updateTutorial() "
    >
        Update
    </button>
    <p>{{ message }}</p>
</div>

<div *ngIf="!currentTutorial.id">
    <br />
    <p>Cannot access this Tutorial...</p>
</div>
</ng-template>

```

## Tutorial

Title

Description

Laravel est un framework web open-source écrit en PHP respectant le principe modèle-vue-contrôleur et entièrement développé en programmation orientée objet.

Status: Pending

[Publish](#)[Delete](#)[Update](#)

Exécutez l'application Angular 16

Nous pouvons exécuter cette application avec la commande : **ng serve --port 8081** .

Si le processus réussit, nous ouvrons le navigateur avec l'URL : <http://localhost:8081/> et vérifions-le.


## conclusion

Nous avons construit le projet Angular 16 : une application fonctionnant avec succès avec l'API Web. Nous pouvons désormais créer, afficher, modifier, supprimer ou rechercher des données de manière propre

# Partie Back-end :

## 1) Créer une application Node.js

Tout d'abord, nous créons un dossier :

 Invite de commandes

```
Microsoft Windows [version 10.0.19045.4046]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\user>cd Desktop

C:\Users\user\Desktop>mkdir nodejs-express-sequelize-postgresql

C:\Users\user\Desktop>cd nodejs-express-sequelize-postgresql

C:\Users\user\Desktop\nodejs-express-sequelize-postgresql>
```

Ensuite, nous initialisons l'application Node.js avec un fichier package.json :

```
C:\Users\user\Desktop\nodejs-express-sequelize-postgresql>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (nodejs-express-sequelize-postgresql)
version: (1.0.0)
description: node js rest api with express js sequelize & postgresQL.
entry point: (index.js) server.js
test command:
git repository:
keywords: node js, express, sequelize, postgresql, rest, api
author: Amine
license: (ISC)
About to write to C:\Users\user\Desktop\nodejs-express-sequelize-postgresql\package.json:
```

Nous devons installer les modules nécessaires : express, sequelize, pg, pg-hstore.

Exécutez la commande : `npm install express sequelize pg pg-hstore cors --save`



```

npm notice
npm notice New minor version of npm available! 10.1.0 -> 10.5.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.5.0
npm notice Run npm install -g npm@10.5.0 to update!
npm notice

C:\Users\user\Desktop\nodejs-express-sequelize-postgresql>npm install express sequelize pg pg-hstore cors --save

added 104 packages, and audited 105 packages in 1m

13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\user\Desktop\nodejs-express-sequelize-postgresql>Exécutez npm install -g npm@10.5.0
'Exécutez' n'est pas reconnu en tant que commande interne
ou externe, un programme exécutable ou un fichier de commandes.

C:\Users\user\Desktop\nodejs-express-sequelize-postgresql>npm install -g npm@10.5.0

added 1 package in 19s

24 packages are looking for funding

```

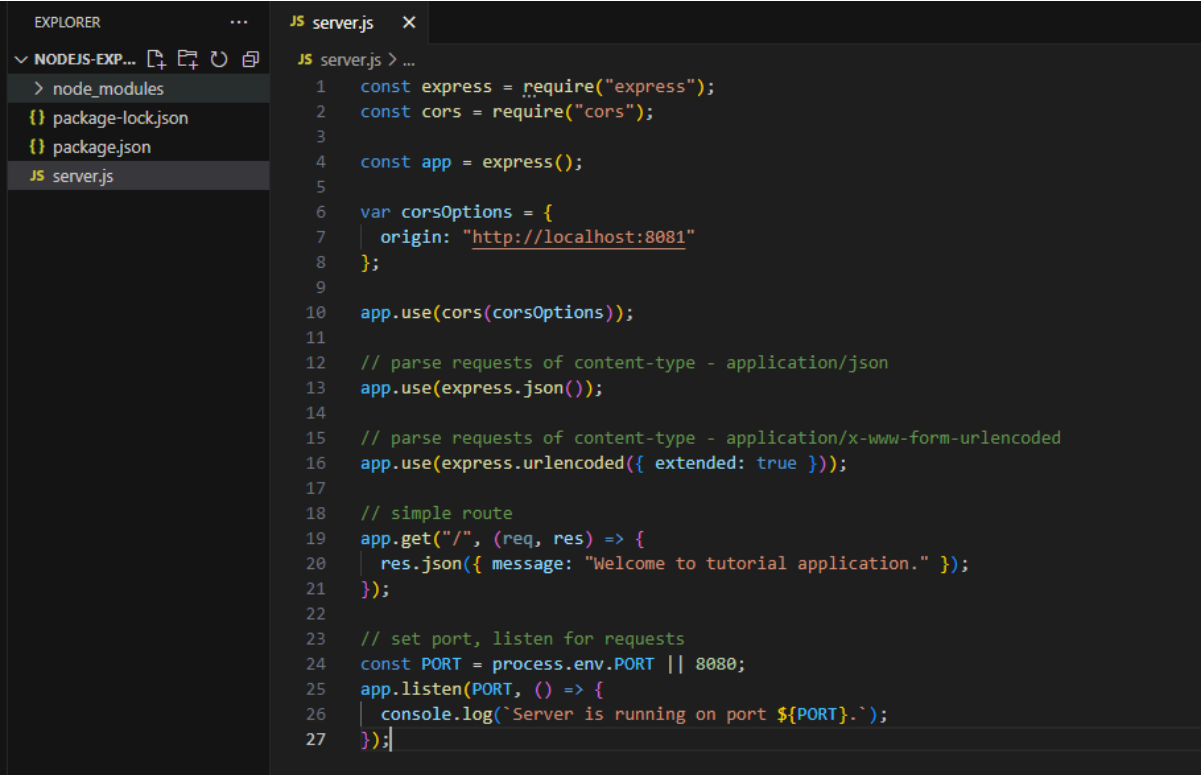
Ci-dessus, il nous a demandé de mettre à jours la version de npm 10.1.0 à 10.5.0

Nous avons utilisé la comande : `npm install -g npm@10.5.0`

\*pg pour PostgreSQL et pg-hstore pour convertir les données au format PostgreSQL hstore.

## 2) Configurer le serveur Web Express

Dans le dossier racine, créons un nouveau fichier server.js :



```

EXPLORER
  > node_modules
  {} package-lock.json
  {} package.json
  JS server.js

JS server.js
1  const express = require("express");
2  const cors = require("cors");
3
4  const app = express();
5
6  var corsOptions = {
7    | origin: "http://localhost:8081"
8  };
9
10 app.use(cors(corsOptions));
11
12 // parse requests of content-type - application/json
13 app.use(express.json());
14
15 // parse requests of content-type - application/x-www-form-urlencoded
16 app.use(express.urlencoded({ extended: true }));
17
18 // simple route
19 app.get("/", (req, res) => {
20   | res.json({ message: "Welcome to tutorial application." });
21 });
22
23 // set port, listen for requests
24 const PORT = process.env.PORT || 8080;
25 app.listen(PORT, () => {
26   | console.log(`Server is running on port ${PORT}.`);
27 });

```

Voici ce que nous faisons :

- Importer les modules express et cors :

Express sert à construire des API REST.

cors fournit un middleware Express pour activer le CORS avec diverses options.

- Créer une application Express :

Nous créons une application Express à l'aide de `express()`.

Ensuite, nous ajoutons les middlewares `body-parser` (pour l'analyse des requêtes JSON et URLencodé) et `cors` en utilisant la méthode `app.use()`.

*\* Notez que nous définissons `origin: http://localhost:8081` pour le paramètre `cors`, ce qui permet les requêtes inter-domaines (CORS) uniquement en provenance de `http://localhost:8081`.*

- Définir une route GET :

Nous définissons une route GET simple à des fins de test. Cela permet d'envoyer des requêtes GET à un point de terminaison spécifique pour vérifier la fonctionnalité de base de l'API.

- Écouter sur le port 8080 :

Nous utilisons la méthode `app.listen(8080, () => { ... })` pour démarrer le serveur de l'API et le faire écouter les requêtes entrantes sur le port 8080.

Lançons maintenant l'application avec la commande : `node server.js`.

```
13 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\user\Desktop\nodejs-express-sequelize-postgresql>Exécutez npm install -g npm@10.5.0
'Exécutez' n'est pas reconnu en tant que commande interne
ou externe, un programme exécutable ou un fichier de commandes.

C:\Users\user\Desktop\nodejs-express-sequelize-postgresql>npm install -g npm@10.5.0

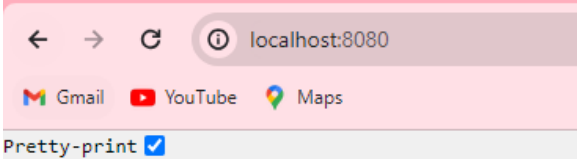
added 1 package in 19s

24 packages are looking for funding
  run `npm fund` for details

C:\Users\user\Desktop\nodejs-express-sequelize-postgresql>code .

C:\Users\user\Desktop\nodejs-express-sequelize-postgresql>node server.js
Server is running on port 8080.
```

Ouvrez votre navigateur avec l'url <http://localhost:8080/>, nous verrons :



```
{
  "message": "Welcome to tutorial application."
}
```

Oui, la première étape est faite. Nous allons travailler avec Sequelize dans la section suivante.

### 3) Configurer la base de données PostgreSQL et Sequelize

Dans le dossier app, nous créons un dossier de configuration séparé pour la configuration avec le fichier db.config.js comme ceci :

```
module.exports = {  
  HOST: "localhost",  
  USER: "postgres",  
  PASSWORD: "admin",  
  DB: "amedb",  
  dialect: "postgres",  
  pool: {  
    max: 5,  
    min: 0,  
    acquire: 30000,  
    idle: 10000  
  }  
};
```

- Les cinq premiers paramètres concernent la connexion PostgreSQL.
- pool est facultatif, il sera utilisé pour la configuration du pool de connexions Sequelize :
  - max : nombre maximum de connexions dans le pool
  - min : nombre minimum de connexions dans le pool
  - idle: durée maximale, en millisecondes, pendant laquelle une connexion peut être inactive avant d'être libérée
  - acquire: durée maximale, en millisecondes, pendant laquelle ce pool tentera d'établir une connexion avant de générer une erreur

## 4)Initialiser Sequelize

Nous allons initialiser Sequelize dans le dossier app/models qui contiendra le modèle à l'étape suivante.

Créez maintenant app/models/index.js avec le code suivant :

```
JS index.js ×
app > models > JS index.js > dbConfig
1  const dbConfig = require("../config/db.config.js");
2
3  const Sequelize = require("sequelize");
4  const sequelize = new Sequelize(dbConfig.DB, dbConfig.USER, dbConfig.PASSWORD, {
5    host: dbConfig.HOST,
6    dialect: dbConfig.dialect,
7    operatorsAliases: false,
8
9    pool: {
10     max: dbConfig.pool.max,
11     min: dbConfig.pool.min,
12     acquire: dbConfig.pool.acquire,
13     idle: dbConfig.pool.idle
14   }
15 });
16
17 const db = {};
18
19 db.Sequelize = Sequelize;
20 db.sequelize = sequelize;
21
22 db.tutorials = require("../tutorial.model.js")(sequelize, Sequelize);
23
24 module.exports = db;
```

Nous n'oubliez pas d'appeler la méthode sync() dans server.js :

```

16 app.use(express.urlencoded({ extended: true }));
17
18 const db = require("../app/models");
19 db.sequelize.sync()
20   .then(() => {
21     console.log("Synced db.");
22   })
23   .catch((err) => {
24     console.log("Failed to sync db: " + err.message);
25   });
26
27 // simple route
28 app.get("/", (req, res) => {
29   res.json({ message: "Welcome to tutorial application." });
30 });
31
32 // set port, listen for requests
33 const PORT = process.env.PORT || 8080;
34 app.listen(PORT, () => {
35   console.log(`Server is running on port ${PORT}.`);
36 });

```

## 5) Définir le modèle Sequelize

Dans le dossier models, j'ai créé le fichier tutorial.model.js comme ceci :

```

1 module.exports = (sequelize, Sequelize) => {
2   ...
3   const Tutorial = sequelize.define("tutorial", {
4     title: {
5       type: Sequelize.STRING
6     },
7     description: {
8       type: Sequelize.TEXT
9     },
10    published: {
11      type: Sequelize.BOOLEAN
12    },
13    imageUrl: {
14      type: Sequelize.STRING
15    }
16  });
17  return Tutorial;
18 };

```

Ce modèle Sequelize représente la table des didacticiels dans la base de données PostgreSQL. Ces colonnes seront générées automatiquement : id, titre, description, publié, image de tutoriel, créé à, mis à jour à.

Après avoir initialisé Sequelize, nous n'avons pas besoin d'écrire des fonctions CRUD, Sequelize les prend toutes en charge :

- créer un nouveau tutoriel : `create(object)`
- trouver un tutoriel par identifiant : `findByPk(id)`
- obtenir tous les tutoriels : `findAll()`
- mettre à jour un didacticiel par id : `update(data, where : { id: id })`
- supprimer un tutoriel : `destroy(where: { id: id })`
- supprimer tous les didacticiels : `détruire (où : {})`
- rechercher tous les didacticiels par titre : `findAll({ où : { title: ... } })`

Ces fonctions seront utilisées dans notre contrôleur.

## 6) Créer le contrôleur

Dans le dossier `app/controllers`, créons `tutorial.controller.js` avec ces fonctions :

- `create`
- `findAll`
- `findOne`
- `update`
- `delete`
- `deleteAll`
- `findAllPublished`

```
const db = require("../models");
const Tutorial = db.tutorials;
const Op = db.Sequelize.Op;

// Create and Save a new Tutorial
exports.create = (req, res) => {
  // Validate request
  if (!req.body.title) {
```

```

    res.status(400).send({
      message: "Content can not be empty!"
    });

    return;
  }

  // Create a Tutorial

  const tutorial = {
    title: req.body.title,
    description: req.body.description,
    published: req.body.published ? req.body.published : false,
    imageUrl: req.body.imageUrl || null
  };

  // Save Tutorial in the database

  Tutorial.create(tutorial)
    .then(data => {
      res.send(data);
    })
    .catch(err => {
      res.status(500).send({
        message:
          err.message || "Some error occurred while creating the
Tutorial."
      });
    });
  };

  // Retrieve all Tutorials from the database.

```

```

exports.findAll = (req, res) => {

  const title = req.query.title;

  var condition = title ? { title: { [Op.iLike]: `%${title}%` } } :
null;

  Tutorial.findAll({ where: condition })

    .then(data => {

      res.send(data);

    })

    .catch(err => {

      res.status(500).send({

        message:

          err.message || "Some error occurred while retrieving
tutorials."

      });

    });

};

// Find a single Tutorial with an id
exports.findOne = (req, res) => {

  const id = req.params.id;

  Tutorial.findByIdPk(id)

    .then(data => {

      if (data) {

        res.send(data);

      } else {

        res.status(404).send({

          message: `Cannot find Tutorial with id=${id}.`

        });

      }

    });

};

```



```

    }

  })

  .catch(err => {

    res.status(500).send({

      message: "Error retrieving Tutorial with id=" + id

    });

  });

};

// Update a Tutorial by the id in the request
exports.update = (req, res) => {

  const id = req.params.id;

  Tutorial.update(req.body, {

    where: { id: id }

  })

  .then(num => {

    if (num == 1) {

      res.send({

        message: "Tutorial was updated successfully."

      });

    } else {

      res.send({

        message: `Cannot update Tutorial with id=${id}. Maybe
Tutorial was not found or req.body is empty!`

      });

    }

  })

  .catch(err => {

```

```

        res.status(500).send({
            message: "Error updating Tutorial with id=" + id
        });
    });
};

// Delete a Tutorial with the specified id in the request
exports.delete = (req, res) => {
    const id = req.params.id;

    Tutorial.destroy({
        where: { id: id }
    })
        .then(num => {
            if (num == 1) {
                res.send({
                    message: "Tutorial was deleted successfully!"
                });
            } else {
                res.send({
                    message: `Cannot delete Tutorial with id=${id}. Maybe
Tutorial was not found!`
                });
            }
        })
        .catch(err => {
            res.status(500).send({
                message: "Could not delete Tutorial with id=" + id
            });
        });
};

```

```

    });

};

// Delete all Tutorials from the database.
exports.deleteAll = (req, res) => {

  Tutorial.destroy({

    where: {},

    truncate: false

  })

  .then(nums => {

    res.send({ message: `${nums} Tutorials were deleted successfully!` });

  })

  .catch(err => {

    res.status(500).send({

      message:

        err.message || "Some error occurred while removing all
tutorials."

    });

  });

};

// Find all published Tutorials
exports.findAllPublished = (req, res) => {

  Tutorial.findAll({ where: { published: true } })

  .then(data => {

    res.send(data);

  })

  .catch(err => {

    res.status(500).send({

```

```
        message:
            err.message || "Some error occurred while retrieving
tutorials."
    });
});
};
```

## 7) Définir des itinéraires (routes)

Lorsqu'un client envoie une requête pour un point de terminaison à l'aide d'une requête HTTP (GET, POST, PUT, DELETE), nous devons déterminer comment le serveur répondra en configurant les routes.

Voici nos itinéraires :

- /api/tutorials: GET, POST, DELETE
- /api/tutorials/:id: GET, PUT, DELETE
- /api/tutorials/published: GET

Créez un `tutorial.routes.js` dans le dossier `app/routes` :

```
JS tutorial.routes.js X
app > routes > JS tutorial.routes.js > ...
1  module.exports = app => {
2      const tutorials = require("../controllers/tutorial.controller.js");
3
4      var router = require("express").Router();
5
6      // Create a new Tutorial
7      router.post("/", tutorials.create);
8
9      // Retrieve all Tutorials
10     router.get("/", tutorials.findAll);
11
12     // Retrieve all published Tutorials
13     router.get("/published", tutorials.findAllPublished);
14
15     // Retrieve a single Tutorial with id
16     router.get("/:id", tutorials.findOne);
17
18     // Update a Tutorial with id
19     router.put("/:id", tutorials.update);
20
21     // Delete a Tutorial with id
22     router.delete("/:id", tutorials.delete);
23
24     // Create a new Tutorial
25     router.delete("/", tutorials.deleteAll);
26
27     app.use('/api/tutorials', router);
28 }
```

Nous devons également inclure les routes dans server.js (juste avant `app.listen()`) :

```

const db = require("../app/models");
db.sequelize.sync()
  .then(() => {
    console.log("Synced db.");
  })
  .catch((err) => {
    console.log("Failed to sync db: " + err.message);
  });

// simple route
app.get("/", (req, res) => {
  res.json({ message: "Welcome to tutorial application." });
});
require("../app/routes/tutorial.routes")(app);
// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});

```

## 8) Testez les API

Exécutions notre application Node.js avec la commande : `nodemon server.js`.  
 ( nodemon est une solution de réexécution automatique au lieu d' exécution manuel chaque fois avec node : [npm i nodemon](#)).

En utilisant Postman, nous allons tester toutes les API ci-dessus.

## a) Créer un nouveau didacticiel à l'aide de l'API POST/tutorials

HTTP <http://localhost:8080/api/tutorials/> Save

POST <http://localhost:8080/api/tutorials/> Send

Params Authorization Headers (8) Body • Pre-request Script Tests Settings Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ☐ Beautify

```
1 {
2   "title": "Gestion de l'État avec Redux",
3   "description": "Apprenez à gérer l'état des applications avec Redux dans vos projets React",
4   "published": true,
5   "imageUrl": "https://www.google.com/imgres?q=image%20upload%20in%20react-redux&imgurl=https%3A%2F%2Fmedia"
6 }
```

Body Cookies Headers (9) Test Results 200 OK 3.51 s 646 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 1,
3   "title": "Gestion de l'État avec Redux",
4   "description": "Apprenez à gérer l'état des applications avec Redux dans vos projets React",
5   "published": true,
6   "imageUrl": "https://www.google.com/imgres?q=image%20upload%20in%20react-redux&imgurl=https%3A%2F%2Fmedia",
7   "updatedAt": "2024-04-06T16:30:16.444Z",
8   "createdAt": "2024-04-06T16:30:16.444Z"
9 }
```

Le résultat dans la base de données :

Servers (1) PostgreSQL 16 aminedb/postgres@PostgreSQL 16

Query Query History No limit

```
12 "createdAt" timestamp with time zone NOT NULL,
13 "updatedAt" timestamp with time zone NOT NULL,
14 CONSTRAINT tutorials_pkey PRIMARY KEY (id)
15 )
16
17 TABLESPACE pg_default;
18
19 ALTER TABLE IF EXISTS public.tutorials
20 OWNER to postgres;
21
22 select * from tutorials;
```

Data Output Messages Notifications

id	title	description
1	Gestion de l'État avec Redux	Apprenez à gérer l'état des applications avec Redux dans vos projets Re...

## b) Récupérez tous les didacticiels à l'aide de l'API GET /tutorials

The screenshot shows a REST client interface with a GET request to `http://localhost:8080/api/tutorials/`. The response is a JSON array of two tutorial objects. The first object has an `id` of 1, and the second has an `id` of 2. Both have the same title and description. The `published` status is `true` for both. The `imageUrl` is a placeholder URL. The `createdAt` and `updatedAt` timestamps are shown for each.

```
1 [
2   {
3     "id": 1,
4     "title": "Gestion de l'État avec Redux",
5     "description": "Apprenez à gérer l'état des applications avec Redux dans vos projets React",
6     "published": true,
7     "imageUrl": "https://www.google.com/imgres?q=image%20upload%20in%20react-redux&imgurl=https%3A%2F%2Fmedia",
8     "createdAt": "2024-04-06T16:30:16.444Z",
9     "updatedAt": "2024-04-06T16:30:16.444Z"
10  },
11  {
12    "id": 2,
13    "title": "Gestion de l'État avec Redux",
14    "description": "Apprenez à gérer l'état des applications avec Redux dans vos projets React",
15    "published": true,
16    "imageUrl": "https://www.google.com/imgres?q=image%20upload%20in%20react-redux&imgurl=https%3A%2F%2Fmedia",
17    "createdAt": "2024-04-06T20:08:10.982Z",
18    "updatedAt": "2024-04-06T20:08:10.982Z"
19  }
20 ]
```

## c) Récupérez un seul tutoriel par identifiant en utilisant GET /tutorials/:id Api

The screenshot shows a REST client interface with a GET request to `http://localhost:8080/api/tutorials/2`. The response is a JSON object representing the tutorial with `id` 2. The `id` field in the response is highlighted with a red box.

```
1 {
2   "id": 2,
3   "title": "Gestion de l'État avec Redux",
4   "description": "Apprenez à gérer l'état des applications avec Redux dans vos projets React",
5   "published": true,
6   "imageUrl": "https://www.google.com/imgres?q=image%20upload%20in%20react-redux&imgurl=https%3A%2F%2Fmedia",
7   "createdAt": "2024-04-06T20:08:10.982Z",
8   "updatedAt": "2024-04-06T20:08:10.982Z"
9 }
```



d) Mettre à jour un tutoriel à l'aide de PUT /tutorials/:id =2

Pour le tutoriel ayant id =2 nous avons modifié le titre et l'état de publication

The screenshot shows a REST client interface with the URL `http://localhost:8080/api/tutorials/2`. The method is set to **PUT**. The request body is a JSON object:

```
{
  "title": "node.js",
  "description": "Apprenez à gérer l'état des applications avec Redux dans vos projets React",
  "published": false,
  "imageUrl": "https://www.google.com/imgres?q=image%20upload%20in%20react-redux&imgurl=https%3A%2F%2Fmedia"
}
```

The fields `"title": "node.js"` and `"published": false` are highlighted with red boxes. The response status is **200 OK** with a message: `"message": "Tutorial was updated successfully."`

e) Recherchez tous les didacticiels publiés à l'aide de GET /tutorials/published Api

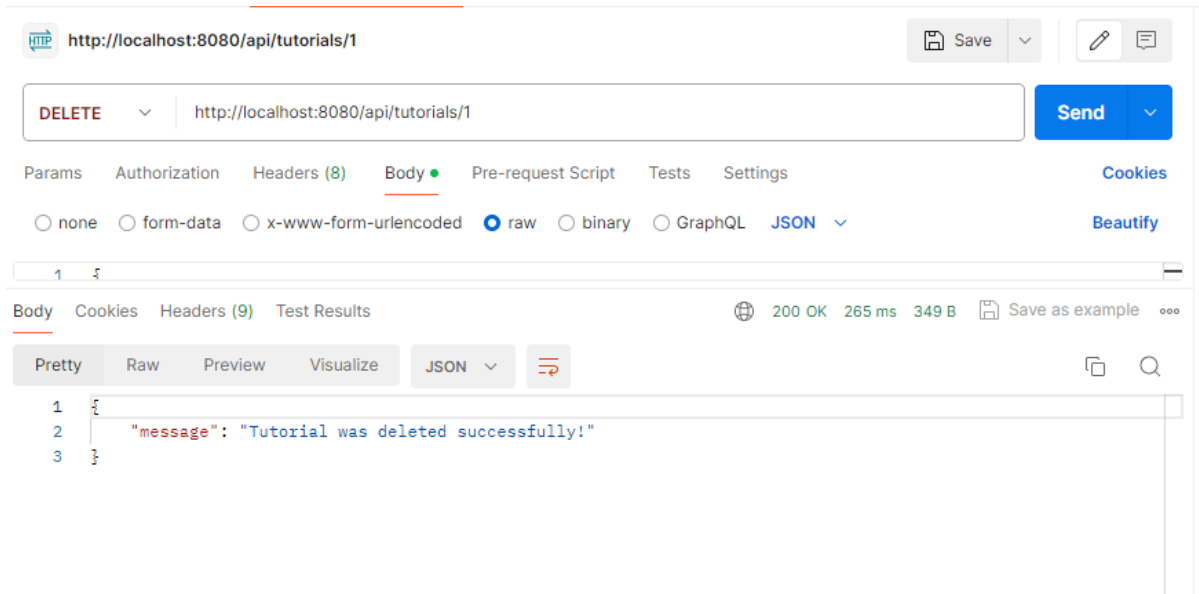
Avec cette fonction nous cherchons juste les tutoriels publiés

The screenshot shows a REST client interface with the URL `http://localhost:8080/api/tutorials/published`. The method is set to **GET**. The response status is **200 OK** with a JSON array of published tutorials:

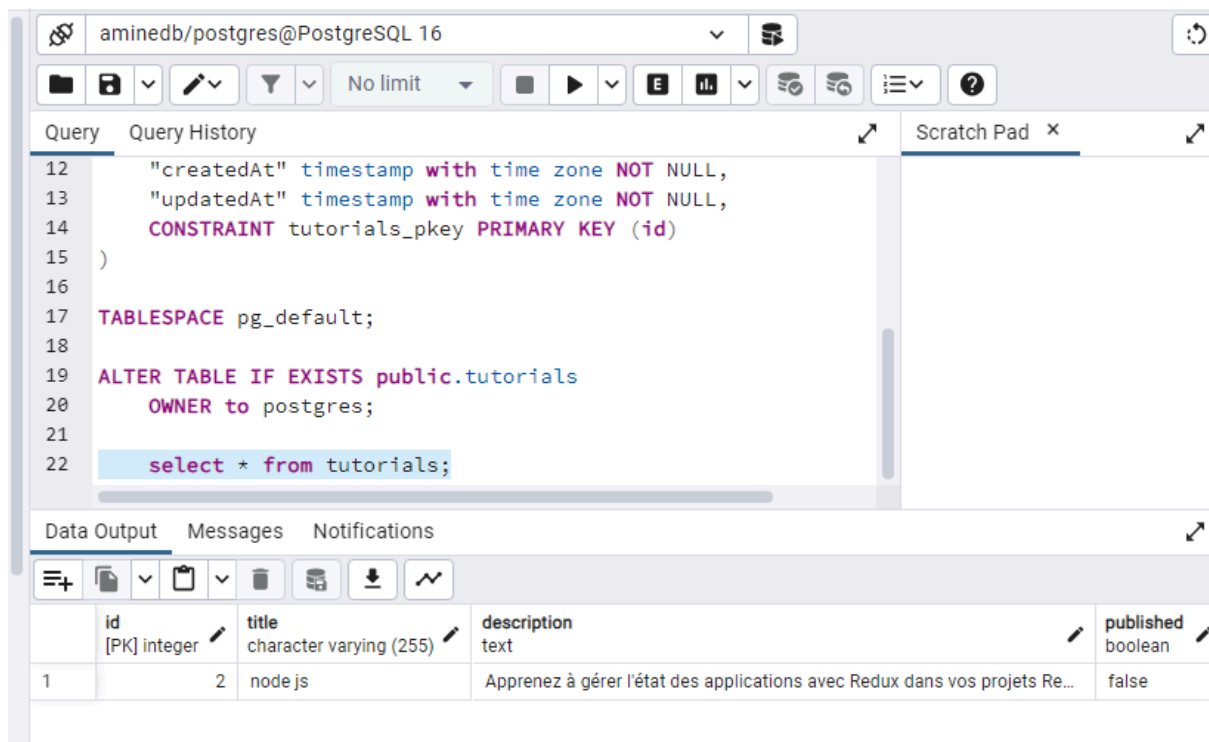
```
[
  {
    "id": 1,
    "title": "Gestion de l'État avec Redux",
    "description": "Apprenez à gérer l'état des applications avec Redux dans vos projets React",
    "published": true,
    "imageUrl": "https://www.google.com/imgres?q=image%20upload%20in%20react-redux&imgurl=https%3A%2F%2Fmedia",
    "createdAt": "2024-04-06T16:30:16.444Z",
    "updatedAt": "2024-04-06T16:30:16.444Z"
  }
]
```

The field `"published": true` is highlighted with a red box.

f) Supprimer un didacticiel à l'aide de DELETE /tutorials/:id  
Api

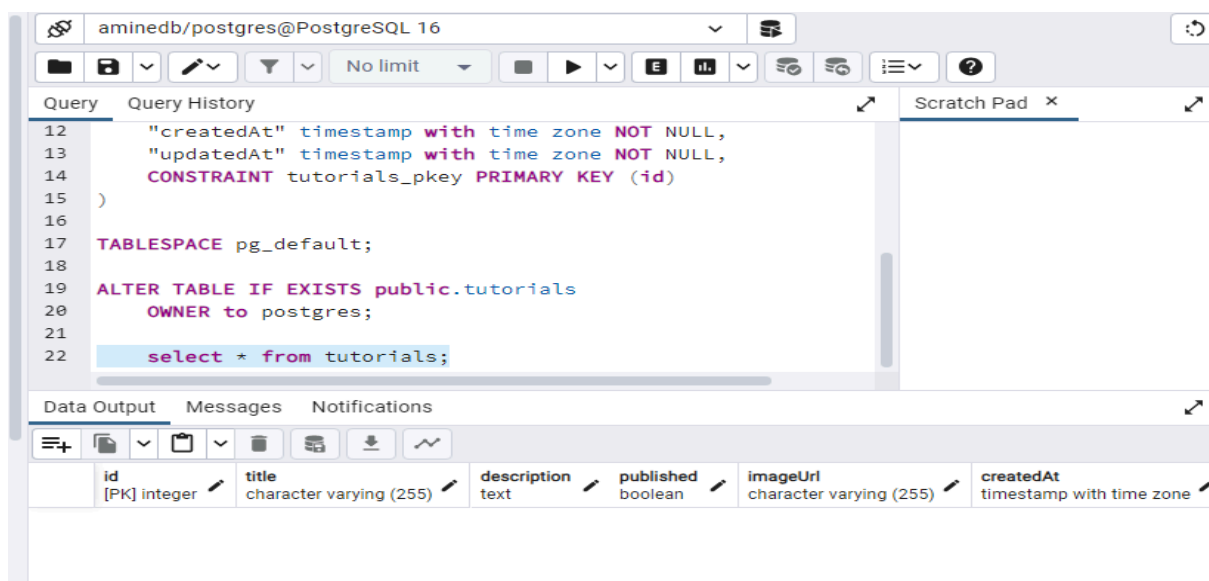
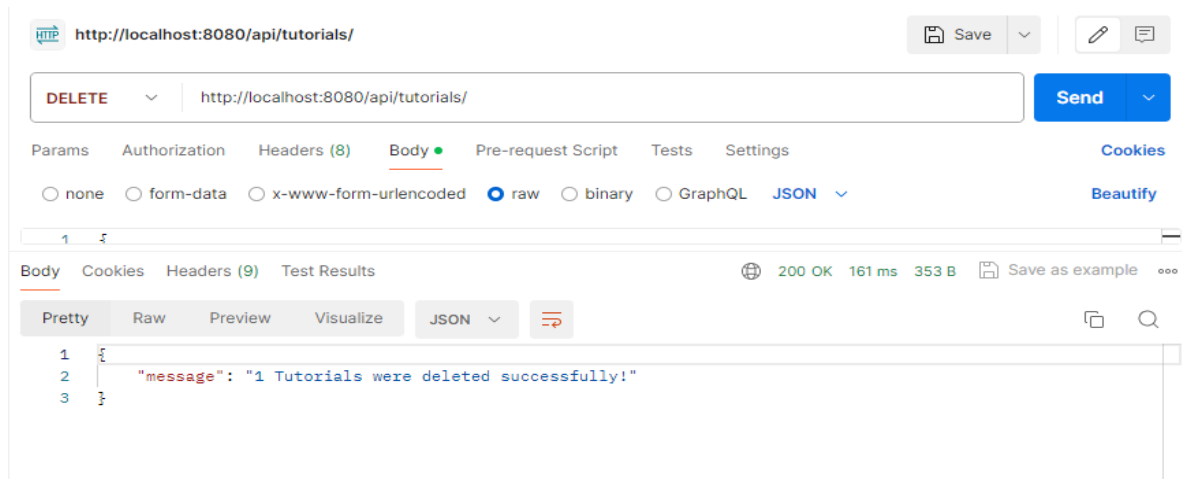


Vérification dans la base que le tutoriel qui a id=1 à été supprimé



## g) Supprimez tous les didacticiels à l'aide de l'API DELETE /tutorials

Tous les tutoriels sont supprimés de la base de données



## 9) Conclusion

Nous avons créé des API Rest Node.js avec un serveur Web Express pour interagir avec la base de données PostgreSQL. Nous savons également comment ajouter une configuration pour la base de données PostgreSQL et Sequelize, créer un modèle Sequelize, écrire un contrôleur et définir des routes pour gérer toutes les opérations CRUD.

