

# First MEAN

## Objectif

Créer un projet avec la stack MEAN  
Créer un serveur avec node et express  
Créer un front end angular  
gérer le cross origin resource sharing

## Angular

Créer un projet angular simplemean

ng new simplemean --no-standalone

```
yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources$ ng new simplemean --no-standalone
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS  [ https://sass-lang.com/documentation/syntax#scss ]
  Sass  [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less  [ http://lesscss.org ]
```

faire enter partout

```
yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources$ ng new simplemean --no-standalone
? Which stylesheet format would you like to use? CSS
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation
(SSG/Prerendering)? (y/N)
```

```

yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources$ ng new simplemean --no-standalone
? Which stylesheet format would you like to use? CSS
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? No
CREATE simplemean/README.md (1064 bytes)
CREATE simplemean/.editorconfig (274 bytes)
CREATE simplemean/.gitignore (548 bytes)
CREATE simplemean/angular.json (2865 bytes)
CREATE simplemean/package.json (1041 bytes)
CREATE simplemean/tsconfig.json (903 bytes)
CREATE simplemean/tsconfig.app.json (263 bytes)
CREATE simplemean/tsconfig.spec.json (273 bytes)
CREATE simplemean/.vscode/extensions.json (130 bytes)
CREATE simplemean/.vscode/launch.json (470 bytes)
CREATE simplemean/.vscode/tasks.json (938 bytes)
CREATE simplemean/src/main.ts (214 bytes)
CREATE simplemean/src/favicon.ico (15086 bytes)
CREATE simplemean/src/index.html (296 bytes)
CREATE simplemean/src/styles.css (80 bytes)
CREATE simplemean/src/app/app-routing.module.ts (245 bytes)
CREATE simplemean/src/app/app.module.ts (393 bytes)
CREATE simplemean/src/app/app.component.css (0 bytes)
CREATE simplemean/src/app/app.component.html (19903 bytes)
CREATE simplemean/src/app/app.component.spec.ts (1065 bytes)
CREATE simplemean/src/app/app.component.ts (211 bytes)
CREATE simplemean/src/assets/.gitkeep (0 bytes)
: Installing packages (npm)...

```

puis aller dans le projet

ouvrir le projet avec vscode

puis creer trois composants comme suit

```

yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources/simplemean$ ng g c posts/post-create
CREATE src/app/posts/post-create/post-create.component.css (0 bytes)
CREATE src/app/posts/post-create/post-create.component.html (26 bytes)
CREATE src/app/posts/post-create/post-create.component.spec.ts (630 bytes)
CREATE src/app/posts/post-create/post-create.component.ts (218 bytes)
UPDATE src/app/app.module.ts (499 bytes)

```

```

yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources/simplemean$ ng g c posts/post-list
CREATE src/app/posts/post-list/post-list.component.css (0 bytes)
CREATE src/app/posts/post-list/post-list.component.html (24 bytes)
CREATE src/app/posts/post-list/post-list.component.spec.ts (616 bytes)
CREATE src/app/posts/post-list/post-list.component.ts (210 bytes)
UPDATE src/app/app.module.ts (597 bytes)

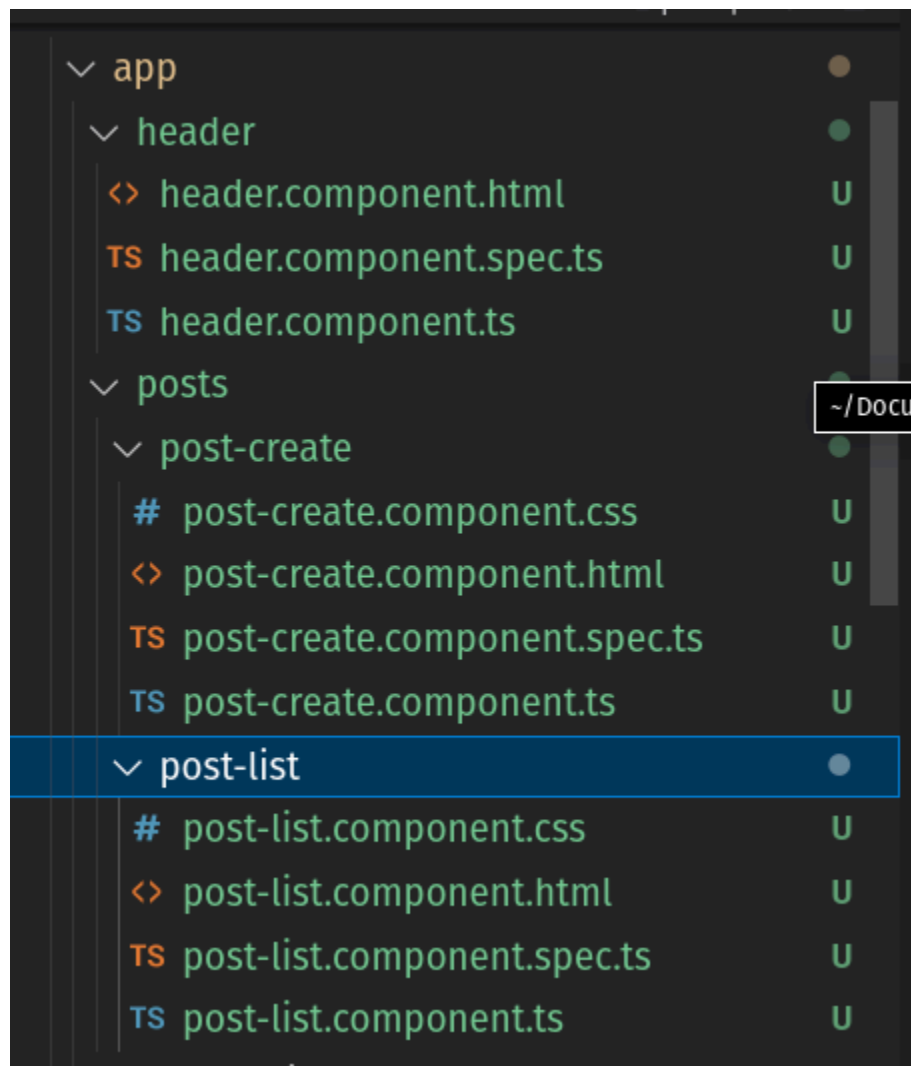
```

```

yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources/simplemean$ ng g c header --inline-style
CREATE src/app/header/header.component.html (21 bytes)
CREATE src/app/header/header.component.spec.ts (601 bytes)
CREATE src/app/header/header.component.ts (175 bytes)
UPDATE src/app/app.module.ts (679 bytes)
yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources/simplemean$ 

```

On obtient ceci



## Installer Material

<https://material.angular.io/>

Qu'est-ce que material?

```
yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources/simplemean$ ng add @angular/material
i Using package manager: npm
✓ Found compatible package version: @angular/material@17.1.2.
✓ Package information loaded.

The package @angular/material@17.1.2 will be installed and executed.
Would you like to proceed? (Y/n) ☐
```

enter

```
✓ Packages successfully installed.
? Choose a prebuilt theme name, or "custom" for a custom theme: (Use arrow keys)
> Indigo/Pink [ Preview: https://material.angular.io?theme=indigo-pink ]
  Deep Purple/Amber [ Preview: https://material.angular.io?theme=deeppurple-amber ]
  Pink/Blue Grey [ Preview: https://material.angular.io?theme=pink-bluegrey ]
  Purple/Green [ Preview: https://material.angular.io?theme=purple-green ]
  Custom
```

enter

```
✓ Packages successfully installed.
? Choose a prebuilt theme name, or "custom" for a custom theme: Indigo/Pink [ Preview: https://material.angular.io?theme=indigo-pink ]
? Set up global Angular Material typography styles? (y/N) ☐
```

inclure les animations

Dans aap.module ajouter les élément de material suivant

```
<> header.component.html U    TS app.module.ts M x
src > app > TS app.module.ts > AppModule
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3
4  import { MatInputModule } from '@angular/material/input';
5  import { MatCardModule } from '@angular/material/card';
6  import { MatButtonModule } from '@angular/material/button';
7  import { MatToolbarModule } from '@angular/material/toolbar';
8  import { MatExpansionModule } from '@angular/material/expansion';
9
10 import { AppRoutingModule } from './app-routing.module';
11 import { AppComponent } from './app.component';
12 import { PostCreateComponent } from './posts/post-create/post-create.component';
13 import { PostListComponent } from './posts/post-list/post-list.component';
14 import { HeaderComponent } from './header/header.component';
15
16 @NgModule({
17   declarations: [
18     AppComponent,
19     PostCreateComponent,
20     PostListComponent,
21     HeaderComponent
22   ],
23   imports: [
24     BrowserModule,
25     AppRoutingModule,
26     MatInputModule,
27     MatCardModule,
28     MatButtonModule,
29     MatToolbarModule,
30     MatExpansionModule
31   ],
32 })
33 export class AppModule {}
```

il faut les importer

Dans header component html

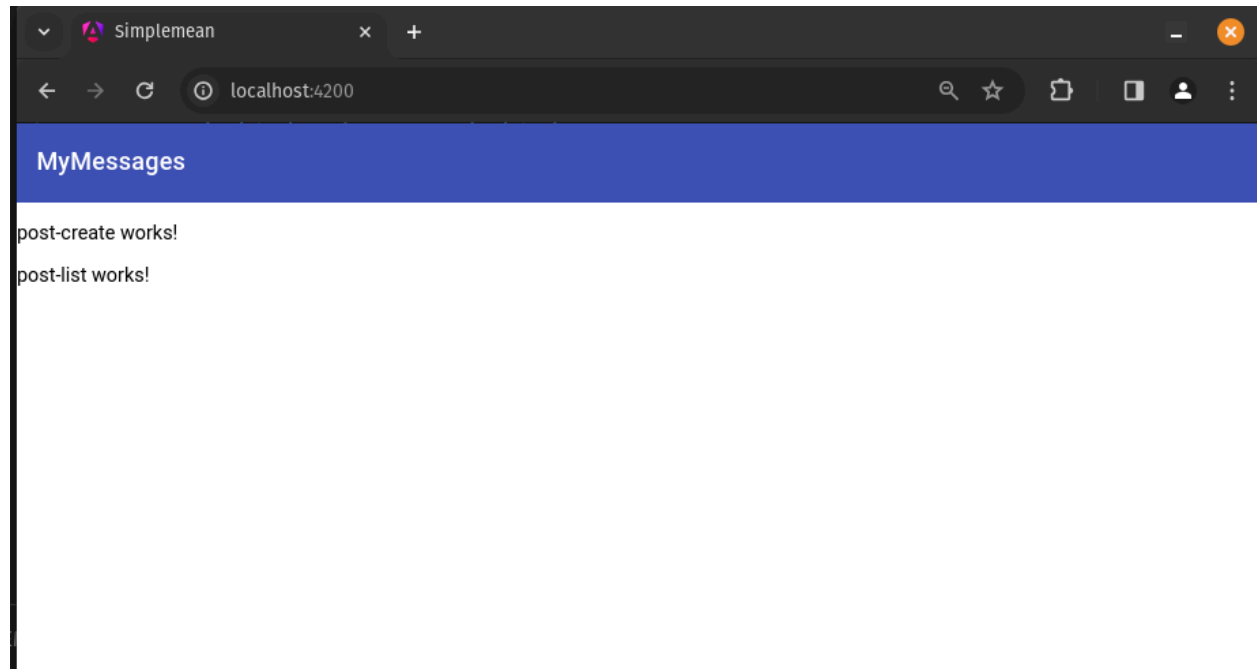
```
JS app.js
> node_modules
  > src
    > app
      > header
        <> header.compone...
        TS header.compone...

Go to component
1 <mat-toolbar color="primary">MyMessages</mat-toolbar>
2
```

Brancher les composant dans app.component.html

```
<> app.component.html x TS app.module.ts JS server.  
src > app > <> app.component.html > main  
Go to component  
1 <app-header></app-header>  
2 <main>  
3   <app-post-create></app-post-create>  
4   <app-post-list></app-post-list>  
5 </main>
```

Faire ng s pour voir



Ajouter dans le module app.module formModule pour les formulaires

```
14 import { HeaderComponent } from '../header/header.component';
15 import { FormsModule } from '@angular/forms';
16
17 @NgModule({
18   declarations: [
19     AppComponent,
20     PostCreateComponent,
21     PostListComponent,
22     HeaderComponent
23   ],
24   imports: [
25     BrowserModule,
26     AppRoutingModule,
27     FormsModule,
28     MatInputModule,
29     MatCardModule,
30     MatButtonModule,

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

## Les template et le style

dans post create css ajouter

```
rs app.module.ts # post-create.component.css ×
src > app > posts > post-create > # post-create.component.css
1  mat-form-field,
2  textarea {
3    width: 100%;
4  }
5
```

Dans post list

```

ts app.module.ts      # post-list.component.css × JS
src > app > posts > post-list > # post-list.component.css >
1  :host {
2      display: block;
3      margin-top: 1rem;
4  }
5
6  .info-text {
7      text-align: center;
8  }
9

```

dans le composant principal

```

src > app > # app.component.css > ...
1  main {
2      width: 80%;
3      margin: 1rem auto;
4  }
5

```

## Create post component

dans post create

créer une methode addPost comme suit



```

2  import { NgForm } from '@angular/forms';
3
4  @Component({
5    selector: 'app-post-create',
6    templateUrl: './post-create.component.html',
7    styleUrls: ['./post-create.component.css']
8  })
9  export class PostCreateComponent {
10    enteredTitle = "";
11    enteredContent = "";
12
13    constructor() {}
14
15    onAddPost(form: NgForm) {
16      if (form.invalid) {
17        return;
18      }
19      console.log(form.value.title);
20
21      form.resetForm();
22    }
23  }
24 }

```

importer NgForm

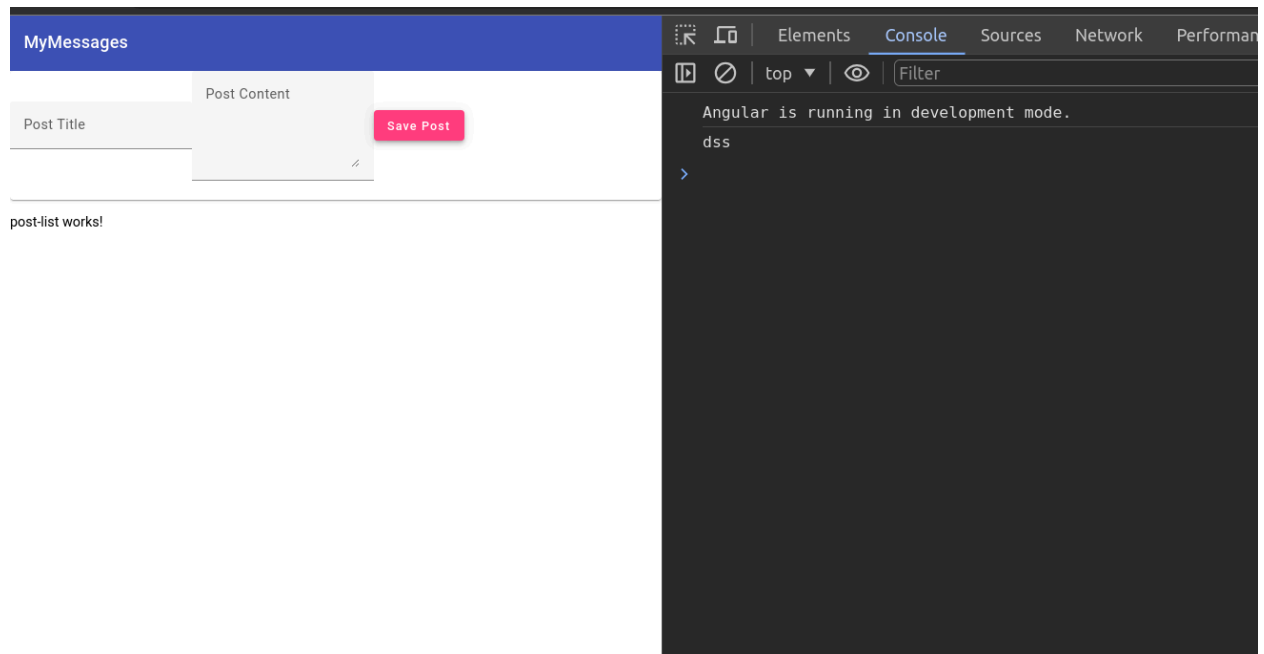
Puis dans le template

```

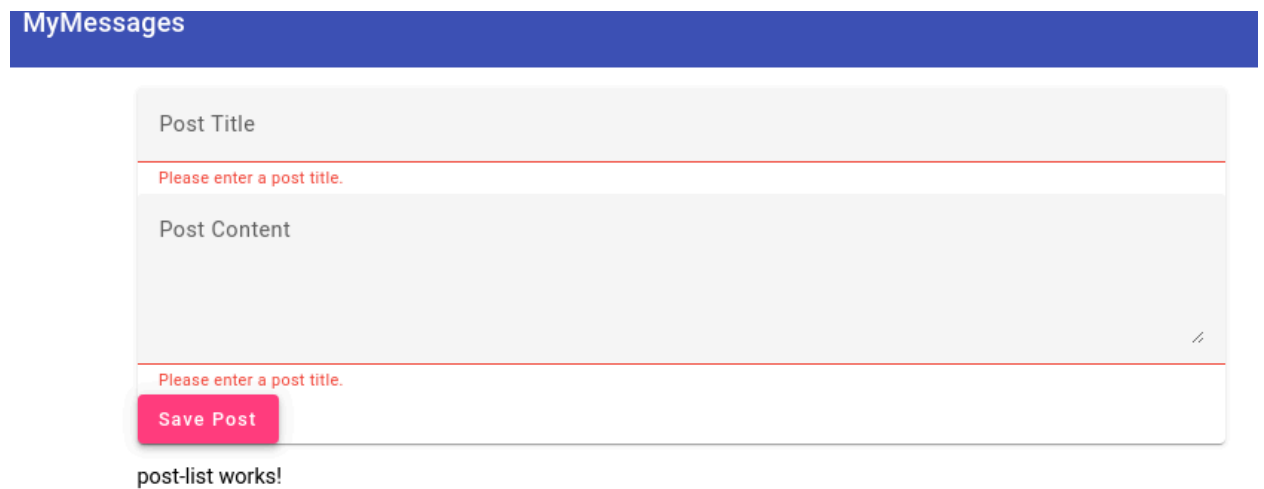
TS post-create.component.ts  <> post-create.component.html
src > app > posts > post-create > <> post-create.component.html > mat-card > form > mat-form-field > input
Go to component
1  <mat-card>
2    <form (submit)="onAddPost(postForm)" #postForm="ngForm">
3      <mat-form-field>
4        <input matInput type="text" name="title" ngModel required minlength="3"
5        placeholder="Post Title" #title="ngModel">
6        @if (title.invalid) {
7          <mat-error>Please enter a post title.</mat-error>
8        }
9      </mat-form-field>
10     <mat-form-field>
11       <textarea matInput rows="4" name="content" ngModel required placeholder="Post Content"
12       #content="ngModel"></textarea>
13       @if (content.invalid){
14         <mat-error>Please enter a post title.</mat-error>
15       }
16     </mat-form-field>
17     <button mat-raised-button color="accent" type="submit">Save Post</button>
18   </form>
19 </mat-card>

```

## Tester



Pause: Expliquons ce code



## Post List Component

Pour pouvoir lister les posts  
Le ts:

```

export class PostListComponent {
  posts = [
    {id:"1", title: "First Post", content: "This is the first post's content" },
    { id:"2", title: "Second Post", content: "This is the second post's content" },
    {id:"3", title: "Third Post", content: "This is the third post's content" }
  ];

  constructor() {}

  ngOnInit() {
    | | console.log(`les posts liste :`, this.posts)
  }

  onDelete(id:string){

  }

  ngOnDestroy() {

  }
}

```

Il est préférable d'implémenter OnInit et OnDestroy qui nous serviront plus tard

## Le template

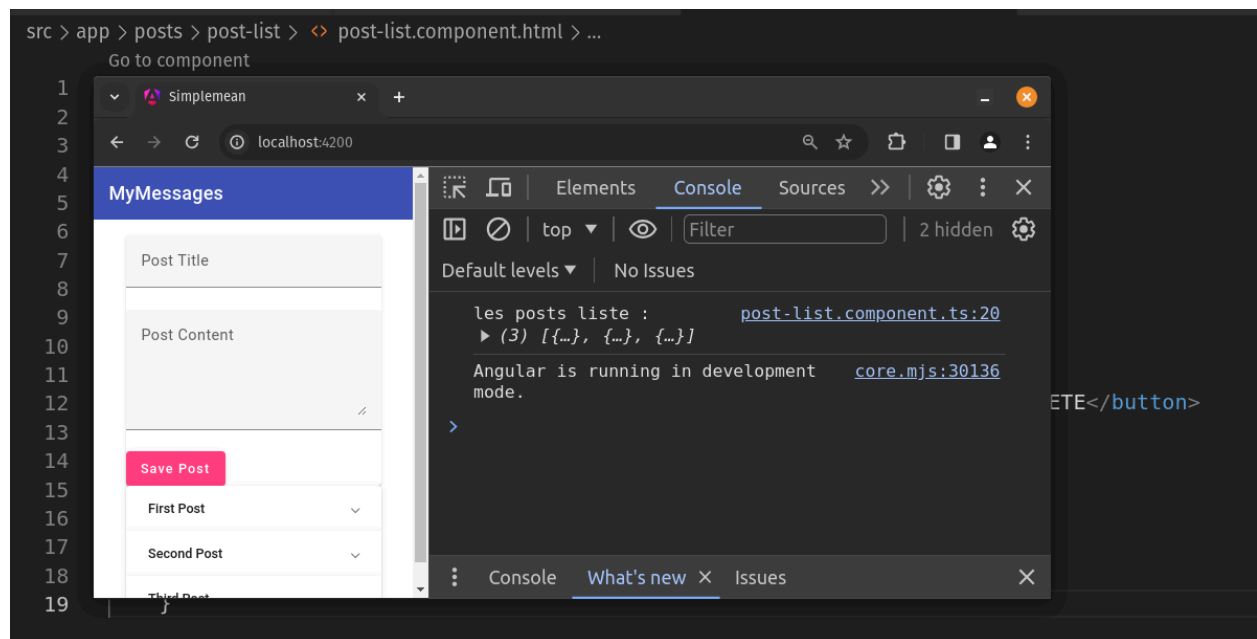
```

src > app > posts > post-list > <> post-list.component.html > ...
Go to component
1  @if (posts.length > 0) {
2    <mat-accordion multi="true">
3
4      @for (post of posts; track $index) {
5        <mat-expansion-panel>
6          <mat-expansion-panel-header>
7            {{ post.title }}
8          </mat-expansion-panel-header>
9          <p>{{ post.content }}</p>
10         <mat-action-row>
11           <button mat-button color="primary">EDIT</button>
12           <button mat-button color="warn" (click)="onDelete(post.id)" >DELETE</button>
13         </mat-action-row>
14       </mat-expansion-panel>
15     }
16   </mat-accordion>
17 }@else {
18   <p class="info-text mat-body-1">No posts added yet!</p>
19 }

```

Que fait ce code?

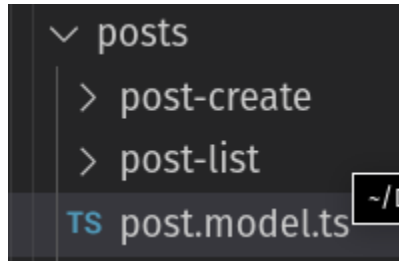
Que sont les directives de structure?



Bravo!

## Le Modele

Dans post, créer un fichier model.ts



Il s'agit d'une interface

```
src > app > posts > TS post.model.ts > Pos
1  export interface Post {
2      id: string;
3      title: string;
4      content: string;
5  }
6
```

Dans post list ts on peut typer notre liste

```
2 import { Post } from "../post.model";
3
4 @Component({
5   selector: 'app-post-list',
6   templateUrl: './post-list.component.html',
7   styleUrls: ['./post-list.component.css']
8 })
9 export class PostListComponent {
10   posts: Post[] = [
11     {id:"1", title: "First Post", content: ""},
12     { id:"2", title: "Second Post", content: ""},
13     {id:"3", title: "Third Post", content: ""}
14   ];
15 }
```

Cela contrôlera le type au cas ou les champs ne seraient pas conformes:

```

2   import { Post } from "../post.model";
3
4   @Component({
5     selector: 'app-post-list',
6     templateUrl: './post-list.component.html',
7     styleUrls: ['./post-list.component.css']
8   })
9   export class PostListComponent {
10    posts:Post[] = [
11      { title: "First Post", content: "This is
12      { id:"2", title: "Second Post", content:
13      {id:"3", title: "Third Post", content: "
14    ];
15

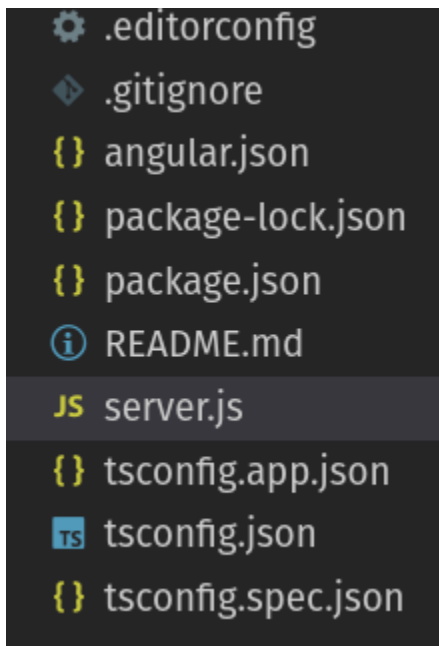
```

Bravo!

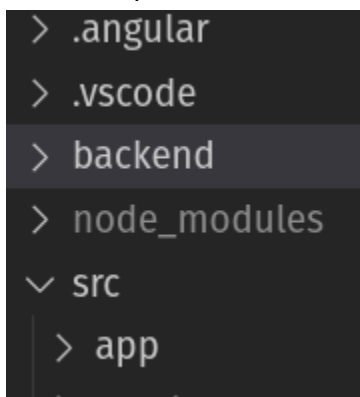
Attaquons nous maintenant au serveur. Nous reviendrons sur le Front pour créer un service qui communique

# Un serveur Rest

A la racine du projet, creer un fichier server.js

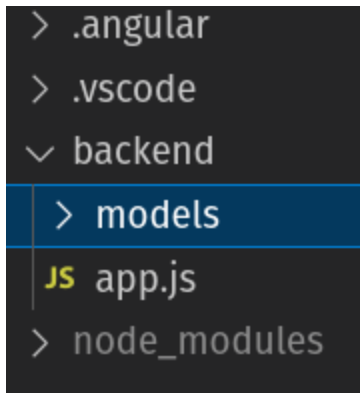


Puis un répertoire backend

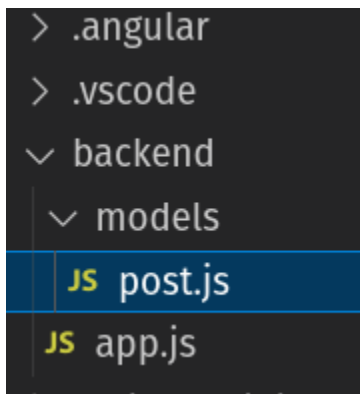


dans ce repertoire un dossier models  
et un fichier app.js





dans models  
un fichier post.js



server.js sera le code bas niveau  
app.js sera notre backend  
et post.js notre "interface" Post mais en javascript et pour notre backend

## Le serveur Node

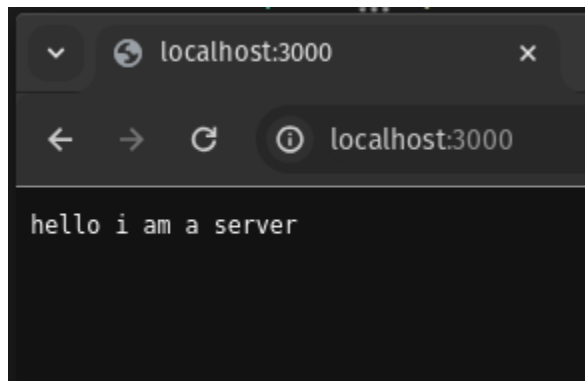
Importer http  
créer le serveur  
écouter le port 3000 par défaut

La méthode `end()` est utilisée pour signaler au serveur que le message de la réponse est complet et que la réponse peut être considérée comme terminée. Cela signifie qu'aucune donnée supplémentaire ne peut être envoyée sur cette réponse après son appel.

**server.js:**

```
1  const http = require("http");
2
3
4  const server = http.createServer(
5    (req, res) => {
6      res.end("hello i am a server");
7    }
8  );
9
10
11  server.listen(process.env.PORT || "3000");
12
13
14
```

tester avec node server.js



# Installation de Express

Installons express avec npm

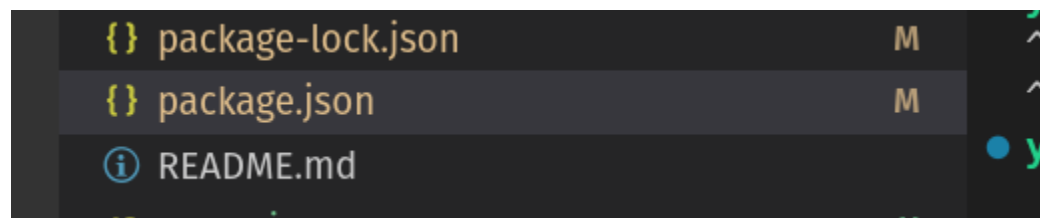
```
yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources/simplemean$ npm install --save express

up to date, audited 973 packages in 2s

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

found 0 vulnerabilities
```

verifier



A screenshot of a file explorer window showing three files: package-lock.json, package.json, and README.md. The package-lock.json and package.json files are highlighted in blue, and the README.md file is highlighted in green. The package-lock.json and package.json files have a yellow icon, and the README.md file has a blue icon.



A screenshot of the package.json file showing the dependencies section. The dependencies are listed as follows: "@angular/platform-browser-dynamic": "^17.1.0", "@angular/router": "^17.1.0", "express": "^4.18.2", "rxjs": "~7.8.0", and "tslib": "^2.3.0". The "express" dependency is highlighted in blue.

Dans app.js

Créons un premier middleware

```
backend > JS app.js > ...
1  const express = require("express");
2  const app = express();
3
4  // middleware
5  app.use((req, res, next) => {
6    console.log('middleware one')
7    next();
8  });
9  app.use((req, res, next) => {
10   res.send("hello from express")
11 });
12
13
14 module.exports = app;
15
```

maintenant dans server.js il faut ajouter notre app

next passe la main au middleware suivante

il faut donc l'appeler tant qu'on ne retourne pas de réponse

JS server.js > ...

```
1  const http = require("http");
2  const app = require("./backend/app");
3
4
5  const server = http.createServer(
6    |   app
7  );
8
9
10 server.listen(process.env.PORT || "3000");
11
12
13
14
```

```
10  
11  
12  
13  
14  
  
PROBLEMS  
  
up to  
119 pa  
run  
  
found 0 vulnerabilities  
yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources/simp  
^C  
yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources/simp  
middleware one  
middleware one  
□
```

Bravo!

## Finalisation du serveur

Un serveur doit gérer beaucoup de cas. Ajoutons ces cas

```

1  const app = require("../backend/app");
2  const debug = require("debug")("node-angular");
3  const http = require("http");
4

```

app : Importe l'application Express configurée depuis le fichier `./backend/app`. Cette application contient toute la logique de routage et les middlewares.

debug : Utilise le module debug pour enregistrer des messages de débogage. Le message de débogage est préfixé par "node-angular", ce qui permet de filtrer les logs spécifiquement pour cette application.

http : Importe le module HTTP natif de Node.js pour créer un serveur HTTP.

```

5  const normalizePort = val => {
6    var port = parseInt(val, 10);
7
8    if (isNaN(port)) {
9      // named pipe
10     return val;
11   }
12
13   if (port >= 0) {
14     // port number
15     return port;
16   }
17
18   return false;
19 };
20

```

On convertit la valeur fournie en un numéro de port valide. Si la valeur n'est pas un nombre, elle pourrait être un "named pipe", donc elle est retournée telle quelle. Si la valeur est un nombre valide et positif, elle est retournée comme le port d'écoute. Sinon, false est retourné, indiquant une erreur.

Un "named pipe" (canal) est un concept informatique utilisé dans les systèmes d'exploitation pour la communication inter-processus (IPC, Inter-Process Communication). Un named pipe permet à deux processus ou plus, qui peuvent ne pas avoir de relation parent/enfant, de communiquer entre eux de manière bidirectionnelle, en utilisant un nom de fichier comme point d'ancrage.

Contrairement aux pipes anonymes (qui sont généralement utilisés pour la communication entre processus parent et enfant), les named pipes existent indépendamment des processus et peuvent être accessibles par n'importe quel processus ayant les permissions appropriées, tant que le système d'exploitation les supportes.

```
20
21  const onError = error => {
22    if (error.syscall !== "listen") {
23      throw error;
24    }
25    const bind = typeof port === "string" ? "pipe " + port : "port " + port;
26    switch (error.code) {
27      case "EACCES":
28        console.error(bind + " requires elevated privileges");
29        process.exit(1);
30        break;
31      case "EADDRINUSE":
32        console.error(bind + " is already in use");
33        process.exit(1);
34        break;
35      default:
36        throw error;
37    }
38  };
39
```

On gère deux types d'erreurs courantes : les erreurs d'accès (EACCES) qui se produisent lorsque le serveur tente d'écouter sur un port pour lequel il n'a pas les privilèges, et les erreurs d'adresse déjà utilisée (EADDRINUSE).



```

39
40   const onListening = () => {
41     const addr = server.address();
42     const bind = typeof port === "string" ? "pipe " + port : "port " + port;
43     debug("Listening on " + bind);
44   };
45
46   const port = normalizePort(process.env.PORT || "3000");
47   app.set("port", port);
48
49   const server = http.createServer(app);
50   server.on("error", onError);
51   server.on("listening", onListening);
52   server.listen(port);

```

Cette fonction est appelée lorsque le serveur commence à écouter sur le port ou le pipe spécifié. Elle utilise le système de débogage configuré.

Le port sur lequel le serveur doit écouter est déterminé en normalisant la valeur de l'environnement PORT ou, par défaut, 3000.

L'application Express app est configurée pour utiliser ce port.

Un serveur HTTP est créé avec app comme gestionnaire pour toutes les requêtes entrantes.

Des gestionnaires d'événements sont ajoutés pour error et listening, afin de gérer respectivement les erreurs de démarrage et de signaler que le serveur écoute.

Enfin, le serveur commence à écouter sur le port configuré.

## Utilisation de npm pour lancer le serveur

Dans package.json ajoutons le raccourci

mais avant ajoutons nodemon

Nodemon est un outil de développement pour Node.js qui surveille automatiquement les modifications apportées aux fichiers dans votre projet et redémarre automatiquement votre serveur Node.js lorsque des changements sont détectés. Cela élimine le besoin de redémarrer manuellement votre serveur à chaque fois que vous faites des modifications.

```
yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources/simplemean$ npm install --save-dev nodemon
added 8 packages, and audited 981 packages in 2s

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

found 0 vulnerabilities
```

```
yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources/simplemean-nostandalone-17$ nodemon server.js
nodemon: command not found
yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources/simplemean-nostandalone-17$
```

```
> .angular 1 {
> .vscode 2   "name": "simplemean-nostandalone-17",
> backend 3   "version": "0.0.0",
  JS app.js 4   "scripts": {
> node_modules 5     "ng": "ng",
> src 6     "start": "ng serve",
  .editorconfig 7     "build": "ng build",
  .gitignore 8     "watch": "ng build --watch --configuration development",
  angular.json 9     "test": "ng test",
  package-lock.json M 10    "start:server": "nodemon server.js"
  package.json M 11  },
  README.md 12  "private": true,
  13  "dependencies": {
```

```
yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources/simplemean-nostandalone-17$ npm run start:server

> simplemean-nostandalone-17@0.0.0 start:server
> nodemon server.js

[nodemon] 3.0.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
```

The screenshot shows a web browser window at localhost:3000 displaying the text "hello from express". Below the browser, the VS Code interface is visible, including a terminal window with the following output:

```
yann@p  
> simp  
> node  
[nodem  
[nodemon] to restart at any time, enter "rs"  
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: js,mjs,cjs,json  
[nodemon] starting `node server.js`  
middleware one  
middleware one  
█
```

changer le message

The screenshot shows a web browser window at localhost:3000 displaying the text "hello,guys from express". Below the browser, the VS Code interface is visible, including a terminal window with the following output:

```
[nodem  
[nodem  
[nodem  
[nodem  
[nodem  
middleware one  
middleware one  
[nodemon] restarting due to changes...  
[nodemon] starting `node server.js`  
middleware one  
middleware one  
█
```

il faut faire F5 cela recharge que le serveur.

## Creation de l'api

Mettons ques données en dur

```
backend > JS app.js > ...
 1  const express = require("express");
 2
 3  const app = express();
 4
 5  app.use("/api/posts", (req, res, next) => {
 6      const posts = [
 7          {
 8              id: "fadf12421l",
 9              title: "First server-side post",
10              content: "This is coming from the server"
11          },
12          {
13              id: "ksajflaj132",
14              title: "Second server-side post",
15              content: "This is coming from the server!"
16          }
17      ];
18      res.status(200).json({
19          message: 'Posts fetched succesfully!',
20          posts: posts
21      });
22  });
23
24  module.exports = app;
25
```

tester dans le navigateur

```
7      {
8        id: "fadf124211"
9      }
10    ]
11  }
12  {"message":"Posts fetched succesfully!","posts":[{"id":"fadf124211","title":"First server-side
13  from the server"},{"id":"ksajflaj132","title":"Second server-side post","content":"This is comi
14
15
16
17
```

PROBLEMS

```
nodem
C
ann@p

simp
node

nodemon] 3.0.3
nodemon] to restart at any time, enter `rs`
nodemon] watching path(s): *.*
nodemon] watching extensions: js,mjs,cjs,json
nodemon] starting `node server.js`
```

## CORS

Il faut ajouter les en-têtes  
ce doit être le premier middleware!

```

4
5
6 app.use((req, res, next) => {
7   res.setHeader("Access-Control-Allow-Origin", "*");
8   res.setHeader(
9     "Access-Control-Allow-Headers",
10    "Origin, X-Requested-With, Content-Type, Accept"
11  );
12  res.setHeader(
13    "Access-Control-Allow-Methods",
14    "GET, POST, PATCH, DELETE, OPTIONS"
15  );
16  next();
17 });
18

```

## Angular HTTP

Essayons de récupérer ces posts

Creons un service

```

TS posts.service.spec.ts  U  [nodemon] starting node server.js
TS posts.service.ts       U  ^C
                                ● yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources/simplemean$ ng g s posts
                                CREATE src/app/posts.service.spec.ts (352 bytes)
                                CREATE src/app/posts.service.ts (134 bytes)
                                yann@pop-os:~/Documents/FORMATIONS/ANGULAR/sources/simplemean$

```

Ajoutons le client HTTP dans module.ts

```

17 import { HttpClientModule } from '@angular/common/http';
18
19 @NgModule({
20   declarations: [
21     AppComponent,
22     PostCreateComponent,
23     HeaderComponent,
24     PostListComponent
25   ],
26   imports: [
27     BrowserModule,
28     AppRoutingModule,
29     FormsModule,
30     BrowserAnimationsModule,
31     HttpClientModule,

```

Pour l'instant créer juste une méthode post dans service et injecter httpClient  
préparer également les deux variables suivantes

```

6 @Injectable({
7   providedIn: 'root'
8 })
9 export class PostsService {
10
11   private posts: Post[] = [];
12   private postsUpdated = new Subject<Post[]>();
13
14   constructor(private http: HttpClient) {}
15
16   getPosts() {}
17
18   {}
19

```

```

15
16     getPosts() {
17         this.http
18             .get<{ message: string; posts: Post[] }>(
19                 "http://localhost:3000/api/posts"
20             )
21             .subscribe(postData => {
22                 console.log('service', postData.posts);
23
24                 this.posts = postData.posts;
25                 this.postsUpdated.next([...this.posts]);
26             });
27     }
28

```

et une méthode pour obtenir les posts mis à jour depuis le service

```

36
37     getPostUpdateListener() {
38         return this.postsUpdated.asObservable();
39     }
40

```

pour ajouter

```

32
33     addPost(title: string, content: string) {
34         console.log('add post');
35
36         const post: Post = { id: 'null', title: title, content: content };
37         this.http
38             .post<{ message: string }>("http://localhost:3000/api/posts", post)
39             .subscribe(data => {
40                 console.log(data.message);
41                 this.posts.push(post);
42                 this.postsUpdated.next([...this.posts]);
43             });
44     }
45

```

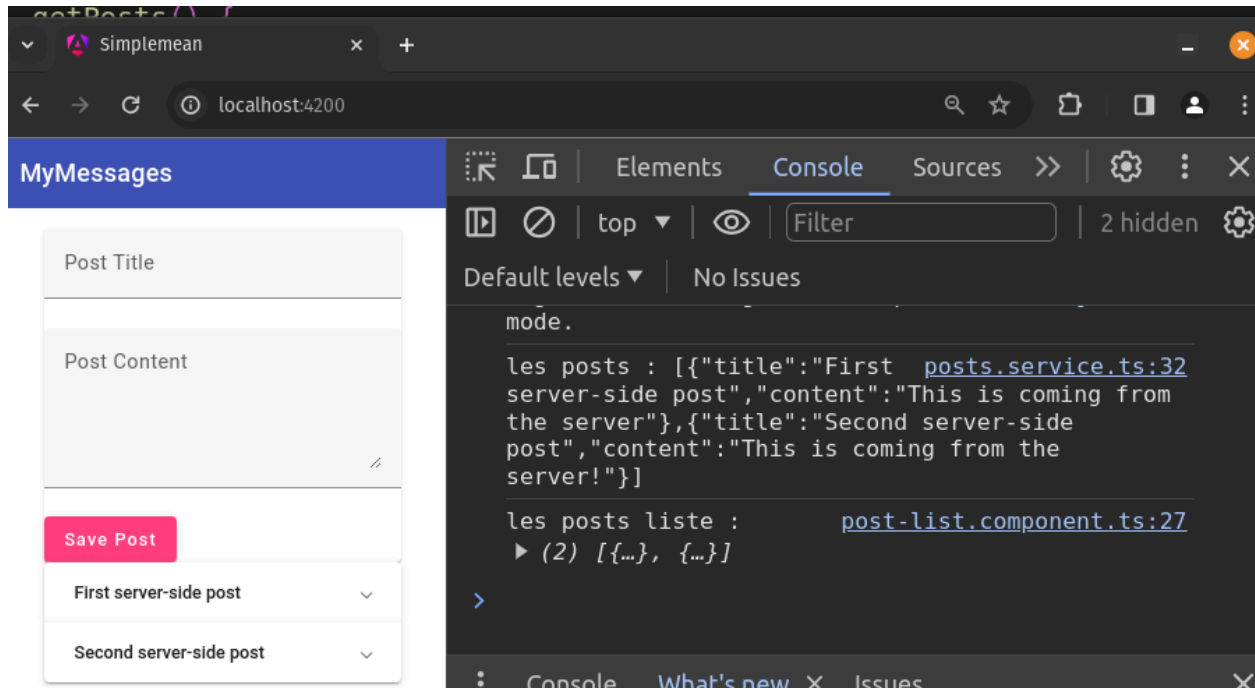
Dans post list .ts



```
src > app > posts > post-list > TS post-list.component.ts > PostListComponent
 9   styleUrls: ['./post-list.component.css']
10 })
11 export class PostListComponent implements OnInit, OnDestroy {
12   // posts = [
13   //   { title: "First Post", content: "This is the first post's content" },
14   //   { title: "Second Post", content: "This is the second post's content" },
15   //   { title: "Third Post", content: "This is the third post's content" }
16   // ];
17   posts: Post[] = [];
18   private postsSub = new Subscription();
19
20   constructor(public postsService: PostsService) {}
21
22   ngOnInit() {
23     this.postsService.getPosts();
24     this.postsSub = this.postsService.getPostUpdateListener()
25       .subscribe((posts: Post[]) => {
26         this.posts = posts;
27         console.log(`les posts liste :`, this.posts);
28       });
29   }
30
31   onDelete(id:string){
32     this.postsService.deletePost(id)
33   }
34
35   ngOnDestroy() {
36     this.postsSub.unsubscribe();
37   }
38 }
39
40
```

on s'inscrit au service

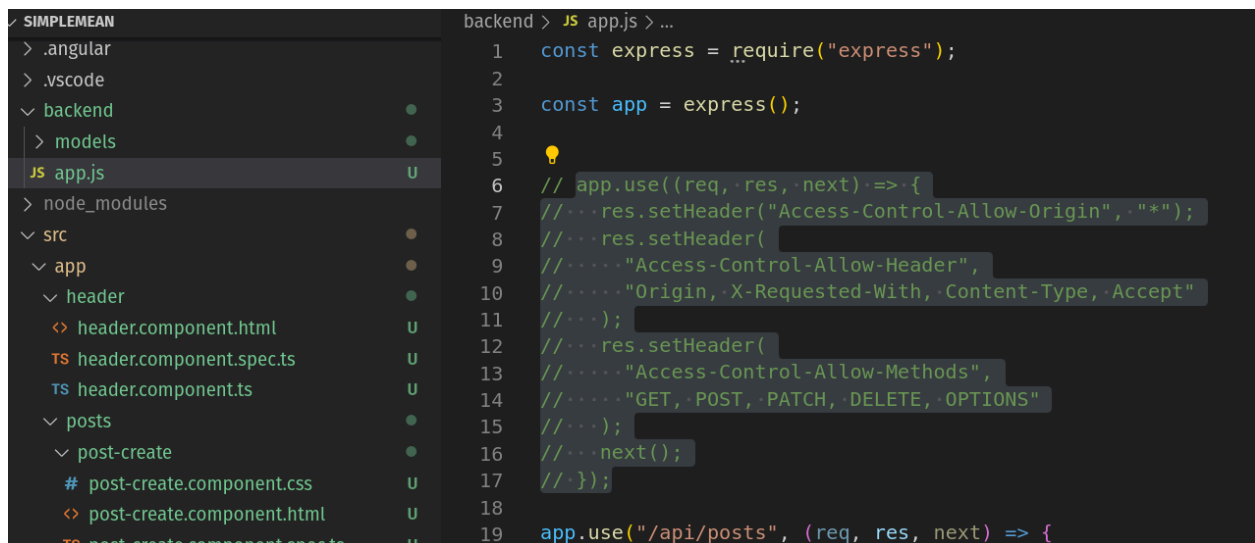
Testons

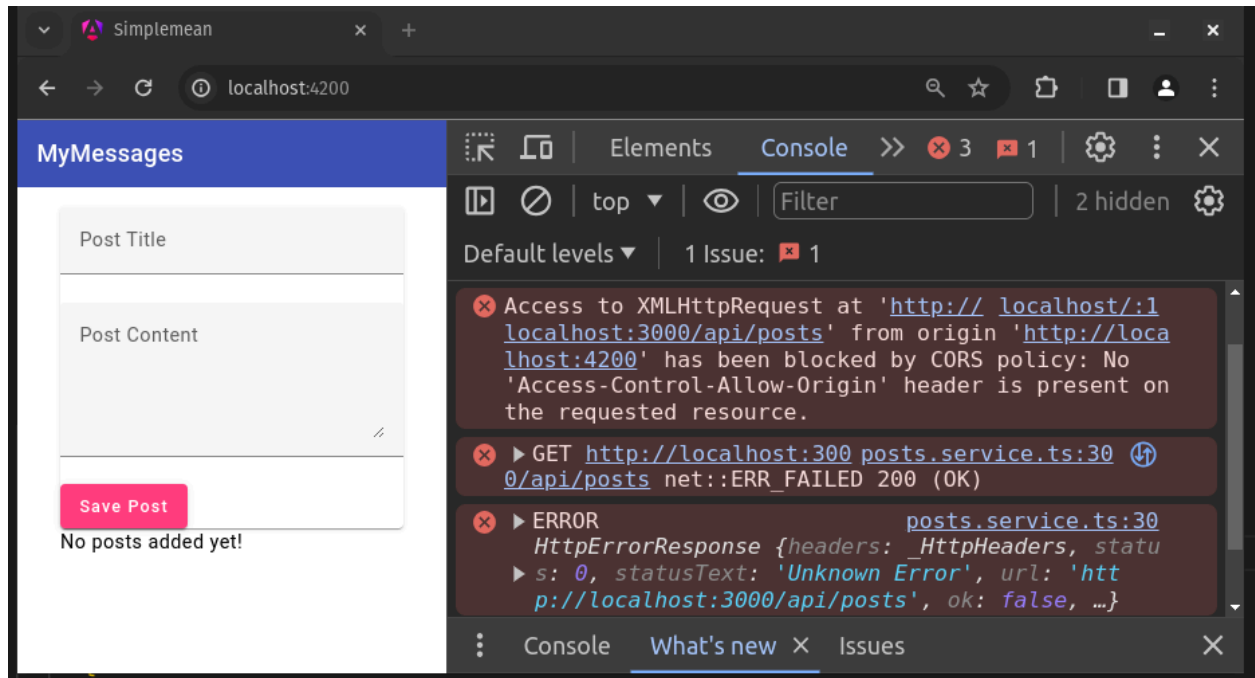


Bravo !

et le CORS ?

Supprimons les en-têtes





L'accès n'est plus autorisé

## Amelioration

`app.use()` peut s'appliquer à toutes les requêtes entrantes, indépendamment de la méthode HTTP (GET, POST, etc.) ou du chemin (URL), à moins qu'un chemin spécifique ne soit fourni comme premier argument.

Méthode HTTP : `app.use()` s'applique à toutes les méthodes HTTP, tandis que `app.get()` `post` `patch` ou `put` s'applique uniquement aux requêtes GET POST PATCH PUT.

Passons le `use` en `get` et ajoutons un endpoint `post`