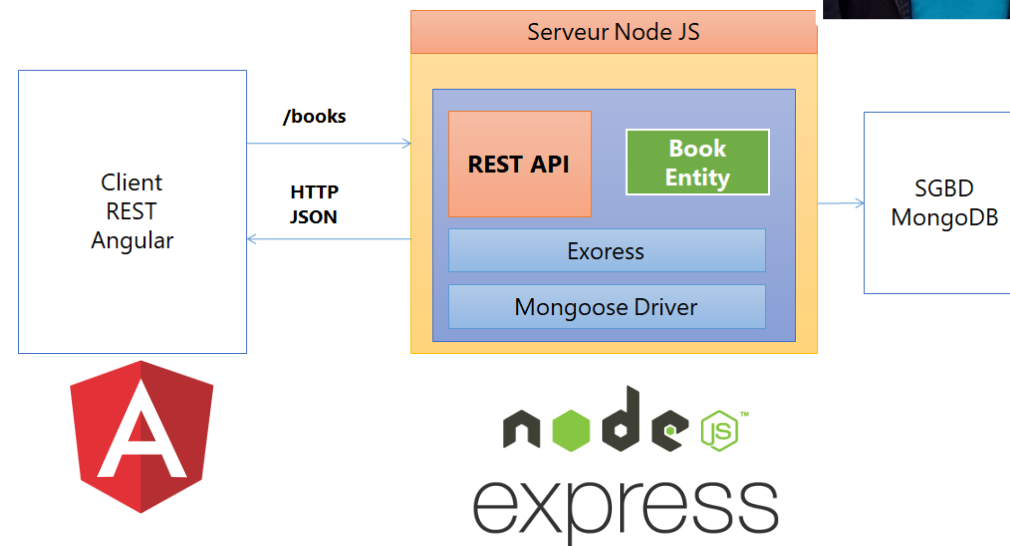
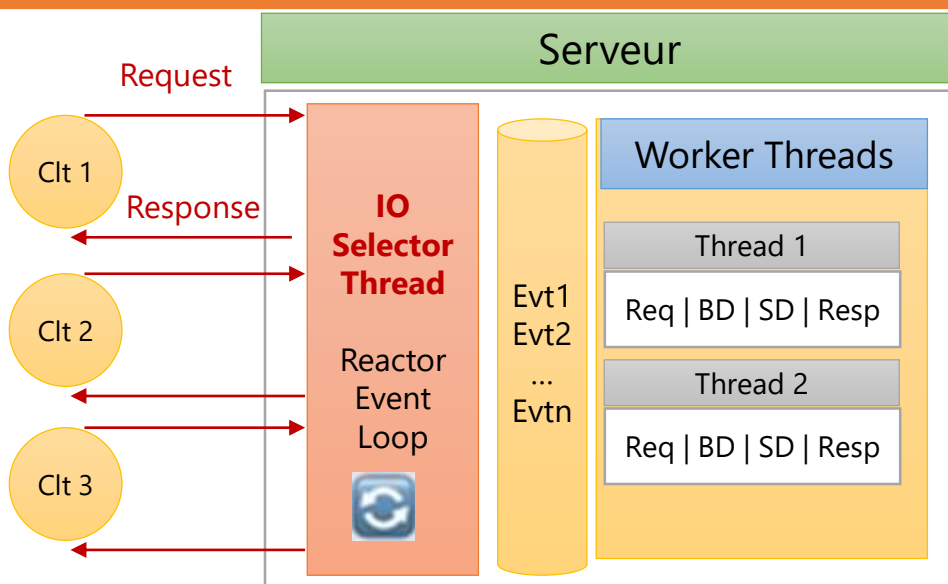


Node JS et Angular avec TypeScript



Mohamed Youssfi

Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)

ENSET, Université Hassan II Casablanca, Maroc

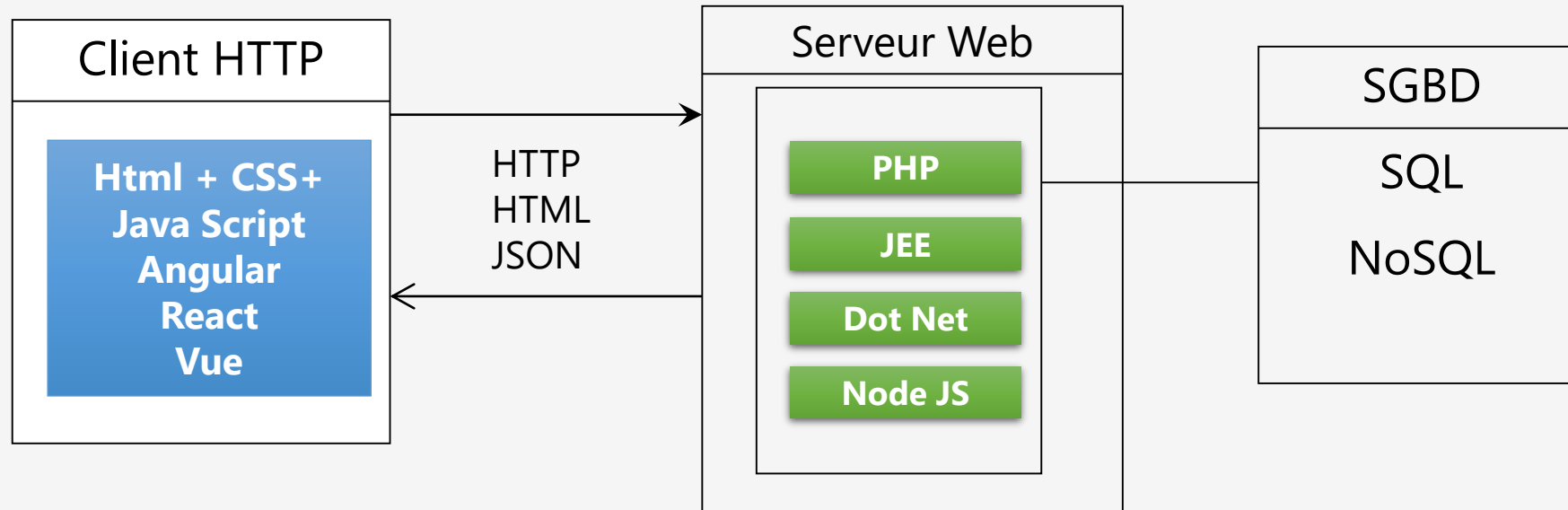
Email : med@youssfi.net

Supports de cours : <http://fr.slideshare.net/mohamedyousfi9>

Chaîne vidéo : <http://youtube.com/mohamedYoussfi>

Recherche : http://www.researchgate.net/profile/Yousfi_Mohamed/publicat

Architecture Web



- Un client web (Browser) communique avec le serveur web (Apache) en utilisant le protocole HTTP
- Une application web se compose de deux parties:
 - La partie **Backend** : S'occupe des traitements effectués coté serveur :
 - Technologies utilisées : PHP, JEE, .Net, Node JS
 - La partie **Frontend** : S'occupe de la présentations des IHM coté Client :
 - Langages utilisés : HTML, CSS, Java Script
- La communication entre la partie Frontend et la partie backend se fait en utilisant le protocole HTTP

Problème de latence

Les applications qui tournent en production

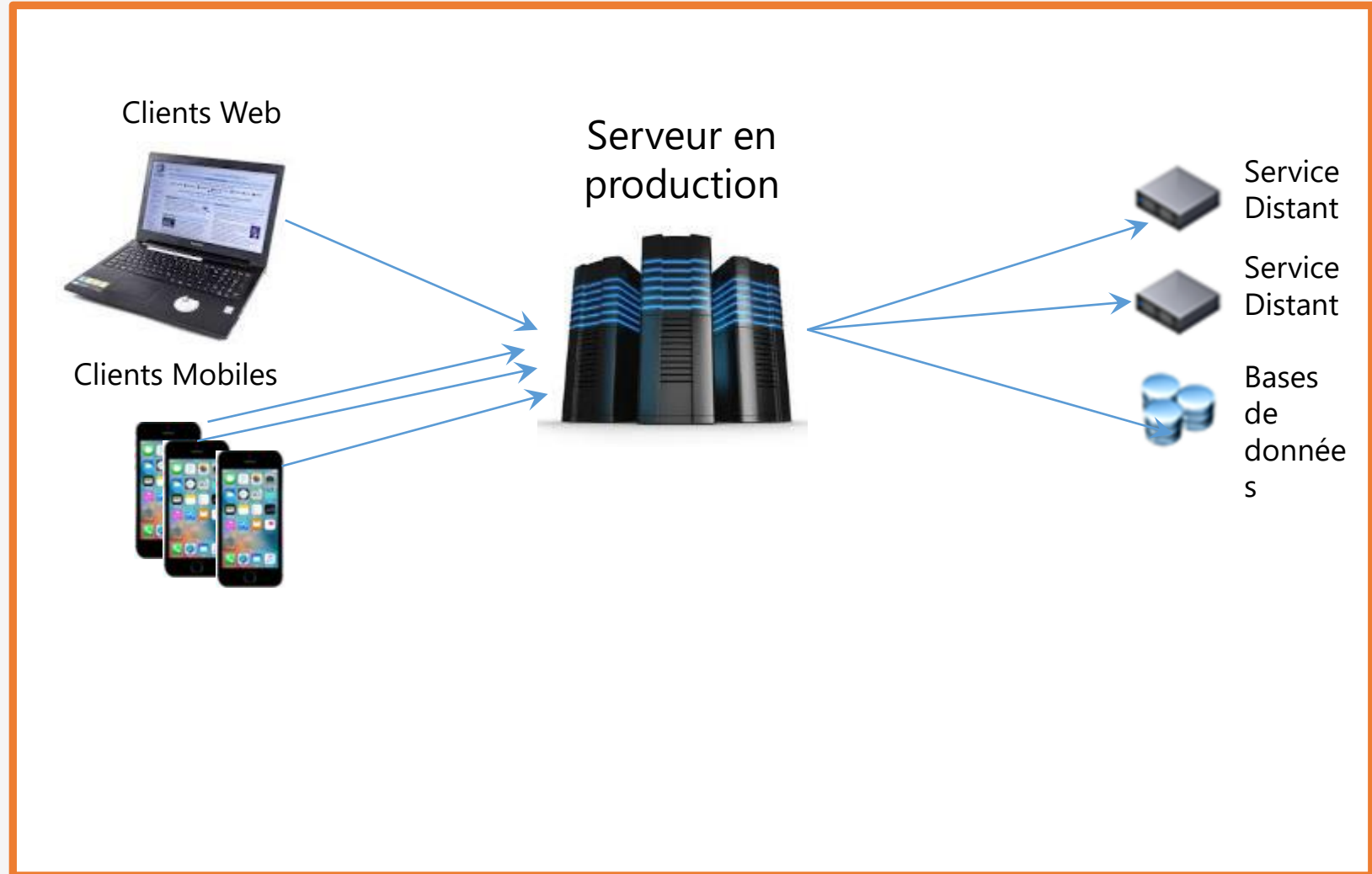
Une variété de clients et **une variété de services distants** qui peuvent être (Bases de données, d'autres services web)

Problème et contraintes :

- Des clients qui ont des connexions lentes (Long lived) et qui monopolisent des ressources sur notre serveur
- Une API distante avec un problème de latence.

Ce qui peut ralentir notre service.

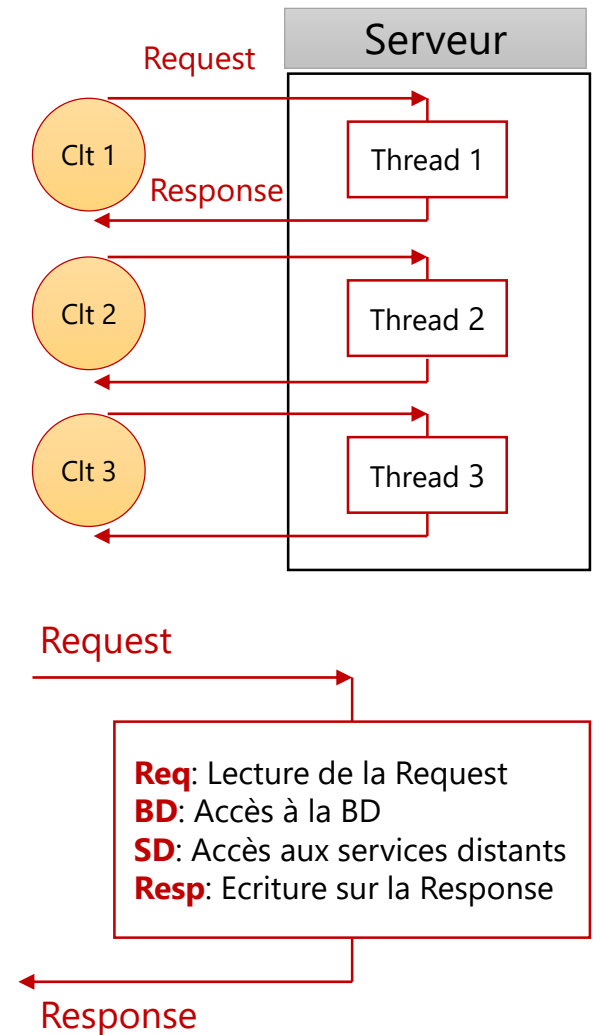
Voir le rendre complètement indisponible



Modèle Multi Threads Bloquant

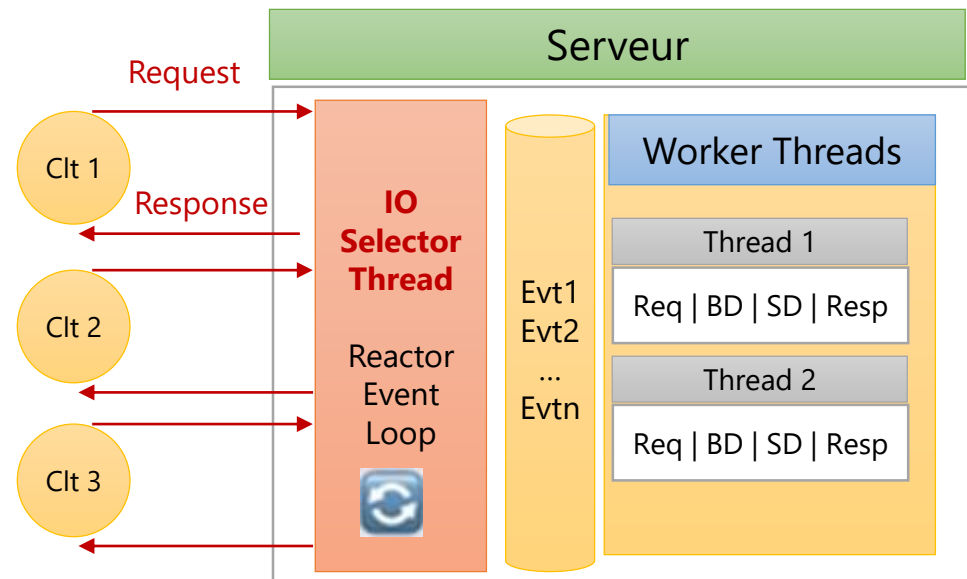
Le modèle classique Bloquant basé sur une Pool de Threads.

- Marche très bien pour de nombreux cas
- A chaque requête, on affecte un Thread tiré du pool de centaines de threads.
- Le rôle de ce thread étant de gérer le traitement de la requête en question
 - Pendant ce traitement on peut avoir :
 1. Lecture des données de la requête
 2. Accéder à une base de données
 3. Accéder à des services distants
 4. Ecriture sur la réponse
 - Toutes ces Entrées Sorties sont bloquantes
 - Le thread attend la lecture et l'écriture sur les IO
 - Dans le cas d'une connexion lente, le thread est mobilisé pour longtemps coté serveur qui empêche d'exploiter les capacités des ressources du serveur.



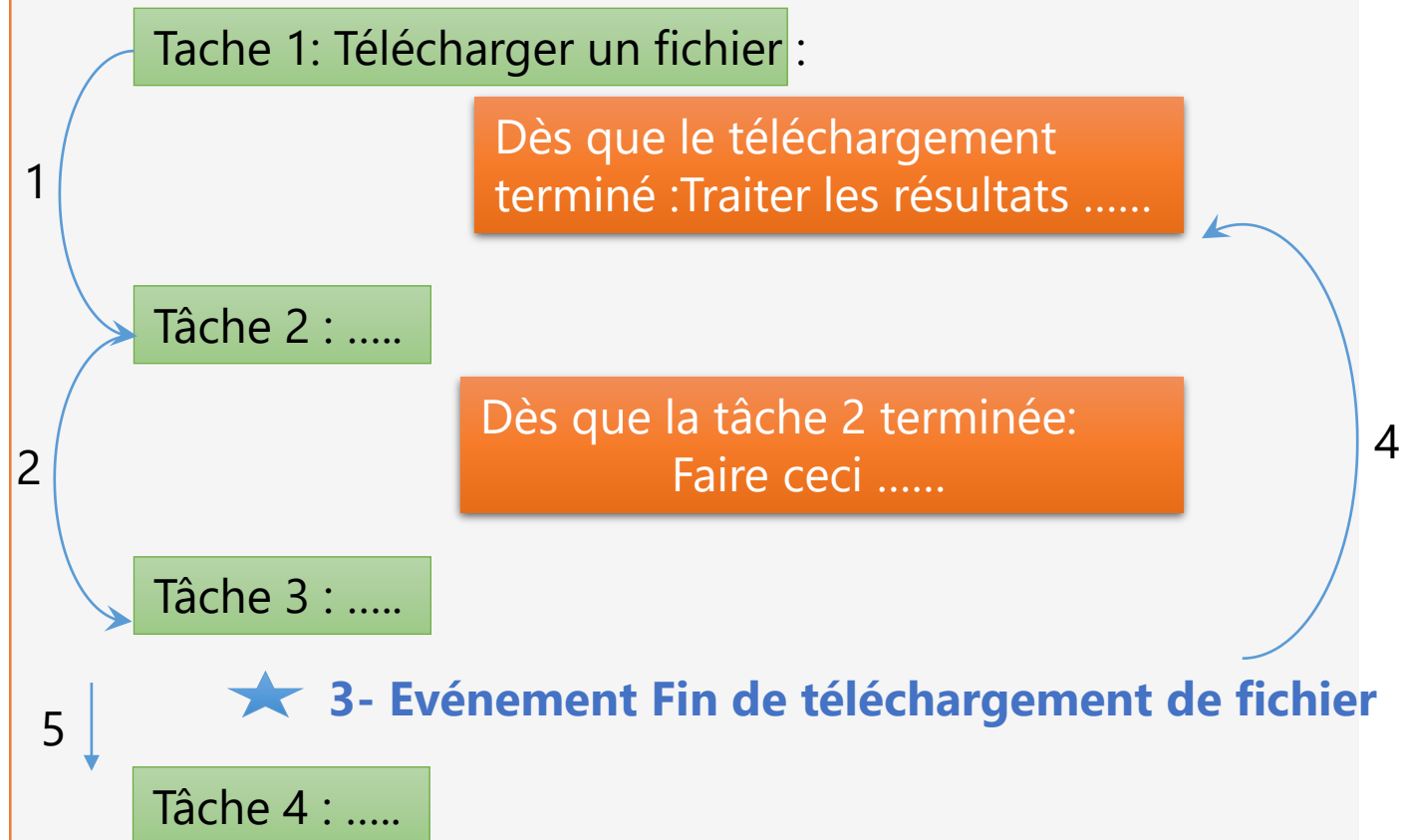
Modèle Single Thread Non Bloquant

- On a un modèle qui utilise un nombre beaucoup plus réduit de threads
 - Un IO Selector Thread dont le rôle est d'orchestrer les entrées sorties Non bloquantes.
 - Cette fois ci tous les IO doivent être fait d'une manière **non bloquantes**. Ce qui fait qu'on va jamais attendre
 - Cet IO thread va **gérer les lectures et les écritures comme des évènements** qu'il va empiler et dépiler dans une Queue d'une manière non bloquante.
 - **Un nombre réduit de Worker Threads** (en fonction du nombre de CPU du serveur)
 - Ces Workers Threads vont s'occuper de traiter les requêtes de manière non bloquantes. Il ne vont jamais attendre. Ils seront toujours entrain de travailler et exploiter aux maximum les ressources du serveur



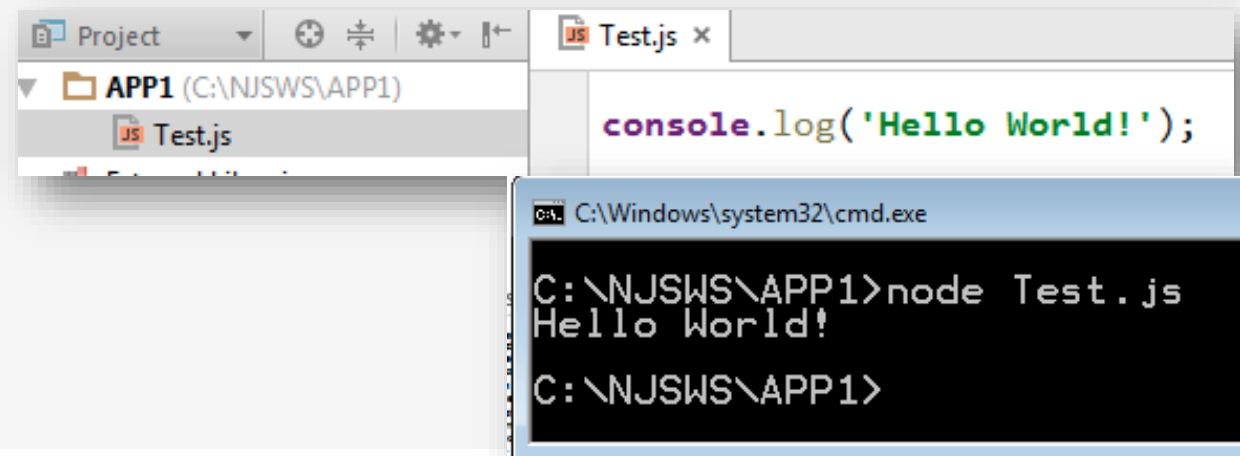
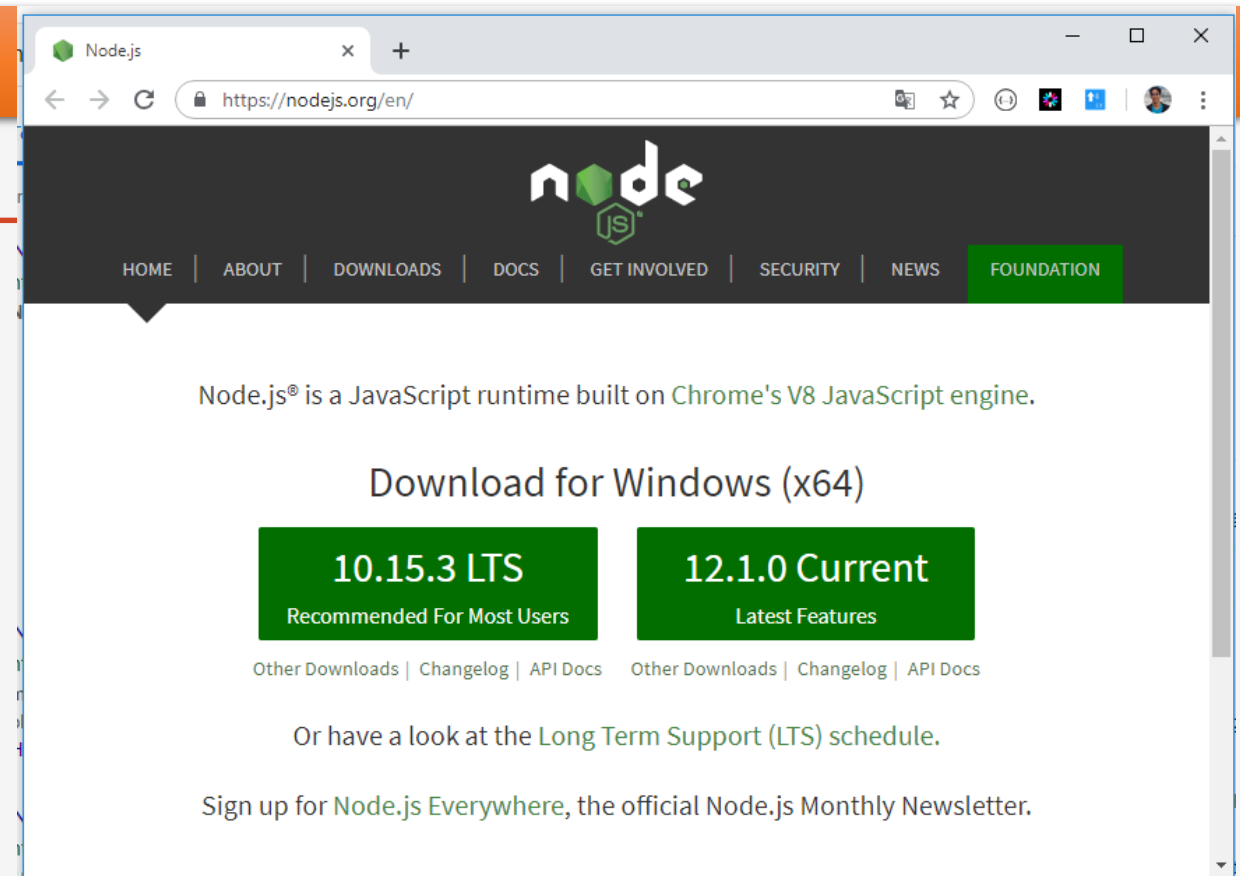
NodeJS

- Node.js est une technologie qui permet de faire du Développement Java Script Coté serveur. Elle se caractérise par :
 - Node n'utilise qu'un seul thread pour gérer les requêtes en utilisant des entrées sorties non bloquantes.
 - Le code NodeJS est asynchrone piloté par des événements et en utilisant des callbacks pour chaque action.
 - Ce qui permet qu'avec un seul Thread, on peut traiter simultanément plusieurs requêtes.
 - Node permet de développer très simplement des applications scalables.
 - S'appuie sur V8, le moteur Javascript de Google utilisé dans Chrome, qui fait partie des moteurs Javascript les plus puissants.



Installation de NodeJS

- Après installation de NodeJS, vous disposez de :
- L'outil **Node** qui permet d'exécuter une application NodeJS
- L'outil **npm** (Node Package Manager) qui est un gestionnaire de paquets officiels de NodeJS.
- Nous utiliserons cet outil pour :
 - Initialiser le projet
 - Installer les dépendances
 - Lancer des scripts
 - Etc..
- Un Projet NodeJS possède un fichier **package.json** pour :
 - Les infos sur le projet
 - Déclarer les dépendances du projet
 - Déclarer les scripts
 - Etc.
- Pour initialiser un projet NodeJS, On utilise la commande :
 - >npm init



Initialisation d'un projet NodeJS

```
> npm init -y
```

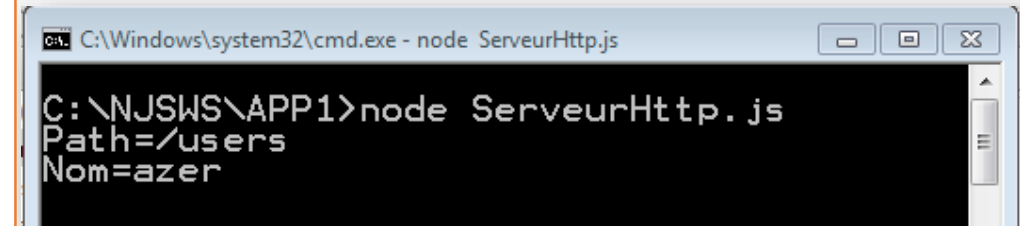
package.json

```
{
  "name": "FirstAppNodeJS",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```


Mise en Place d'un serveur NodeJS

```
/* Inclure le module interne http pour la création du serveur HTTP */
var http=require ('http');
/* Inclure le module interne url pour la récupération des informations de l'url */
var url=require('url');
/* Inclure le module interne querystring pour la récupération des paramètres de l'url */
var querystring=require('querystring');
/* Création du serveur HTTP */
var httpServer=http.createServer(function(request,response){
    // Pour récupérer le path de l'url
    var path=url.parse(request.url).pathname;
    console.log('Path='+path);
    // Pour récupérer les paramètres de l'url
    var params=querystring.parse(url.parse(request.url).query);
    var nom=params['nom'];
    console.log('Nom='+nom);
    // Définir les entêtes de la réponse HTTP
    response.writeHead(200,{ 'content-type': 'text/html' });
    // Envoyer le contenu html dans le corps de la réponse HTTP
    response.end('<h3>Node JS Server, Votre nom est :'+nom+'</h3>');
});
// Démarrer le serveur HTTP en écoutant le port 8000
httpServer.listen(8000);
```

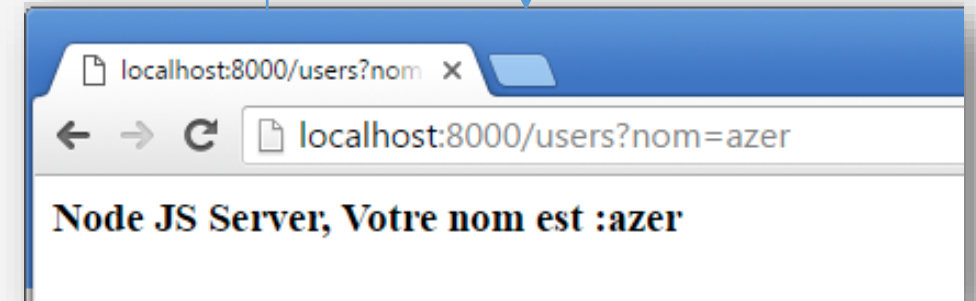
Serveur Node JS



```
C:\Windows\system32\cmd.exe - node ServeurHttp.js
C:\NJSWS\APP1>node ServeurHttp.js
Path=/users
Nom=azer
```

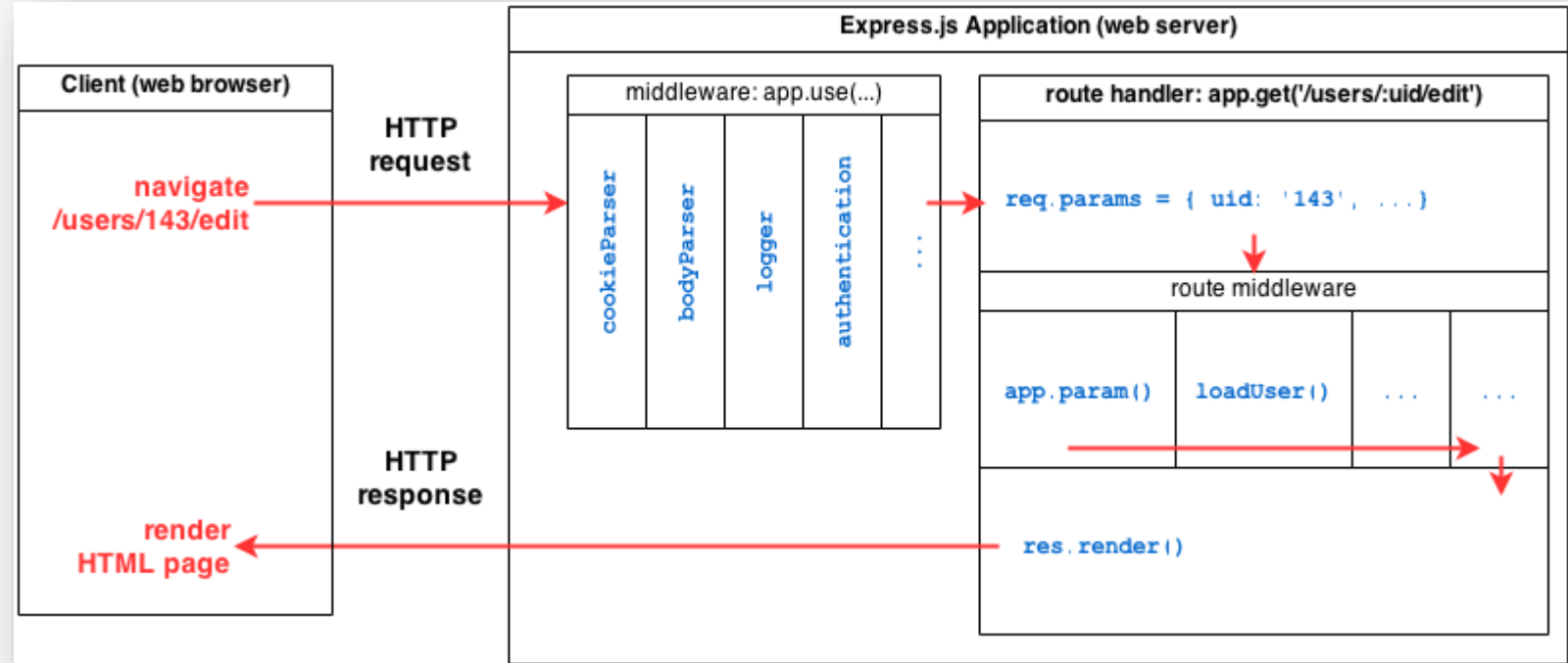
HTTP

Client HTTP



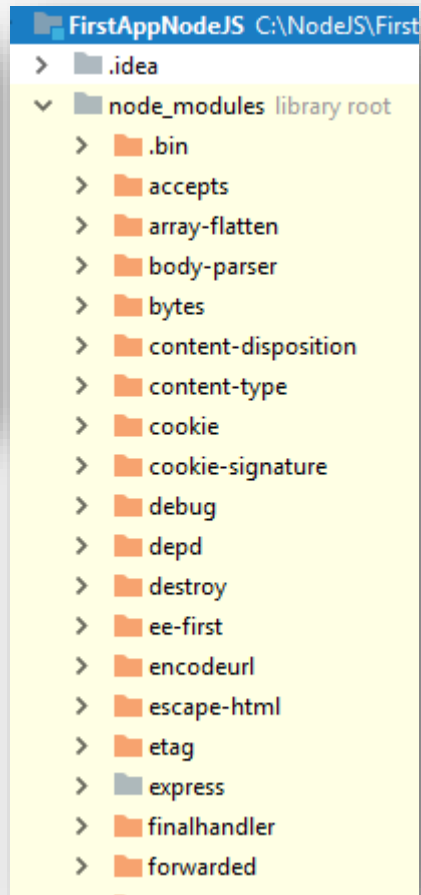
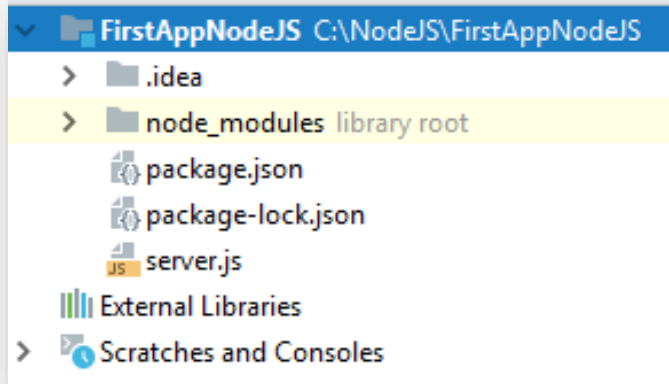
Framework Express

- Express.js est un micro-Framework pour Node.js.
- Il vous fournit des outils de base pour aller plus vite dans la création d'applications Node.js.
- Express offre des fonctionnalités pour :
 - La gestion des routes (système de navigation)
 - Un moteur de Templates (Les vues de l'application)
 - Les middlewares



Installation de Express

- `npm install --save express`



```
{
  "name": "FirstAppNodeJS",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

Gestion des routes avec express

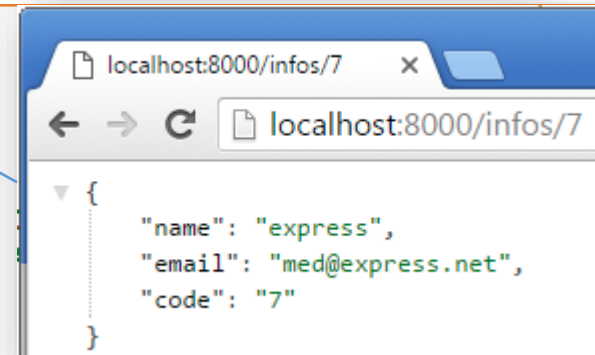
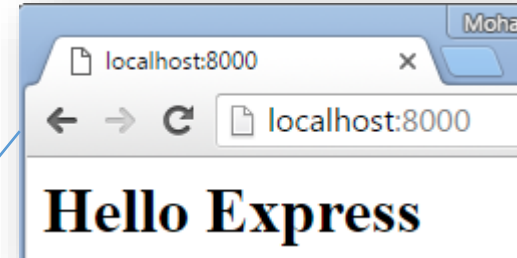
```
const express=require("express");
const app=express();
app.get('/',(req,res)=>{
  res.setHeader('content-type','text/html');
  res.send('<h1>Hello Express</h1>')
});
app.get('/infos/:code',(req,res)=>{
  res.setHeader('content-type','application/json');
  var infos={name:'Express',email:'med@yousfsi.net', code:req.params.code};
  res.end(JSON.stringify(infos));
});
app.listen(7000,()=>{
  console.log('Server Started ..');
});
```

Route statique

Route dynamique

Serveur Node JS

```
C:\NodeJS\FirstAppNodeJS>node server-express
Server Started ..
```



Node JS avec Type Script

- Type Script est langage structure orienté objet qui permet d'écrire facilement des applications java script.
 - Type Script offre entre autres les possibilités de :
 - déclarer les types de variables,
 - Créer des classes et des interfaces
 - Utiliser des décorateurs (Annotations)
 - Utiliser la généricité
 - Etc..
 - Une applications type script est compilée en Java Script qui sera par la suite exécutée par NodeJS ou les browsers web.
- Pour travailler les application NodeJS en Type script, il faudrait installer les dépendances suivantes:
 - **typescript** : Langage Type Script
 - **@types/node** : Fichier de définition typescript pour nodejs
 - **nodemon** : Un utilitaire qui surveillera toute modification du code source et redémarrera automatiquement votre serveur.
 - **concurrently** : un outil pour exécuter plusieurs commandes simultanément
- ```
> npm install --save-dev
typescript nodemon @types/node
concurrently
```

# Node JS avec Type Script

---

```
> npm install --save-dev typescript
nodemon @types/node concurrently
```

## package.json

```
"dependencies": {
 "express": "^4.17.1"
},
"devDependencies": {
 "@types/node": "^12.0.4",
 "concurrently": "^4.1.0",
 "nodemon": "^1.19.1",
 "typescript": "^3.5.1"
}
```

# Node JS avec Type Script

- Premier exemple : index.ts

```
console.log('Hello');
```

- Compiler index.ts

```
> tsc
```

ou

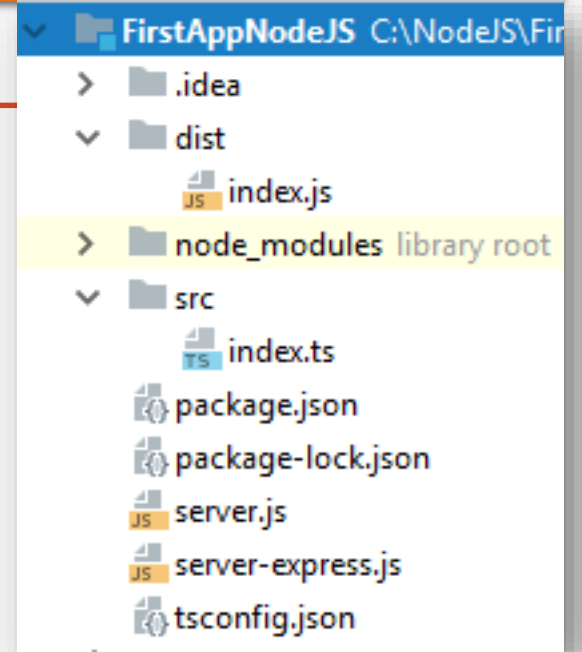
```
> npx tsc
```

- Exécuter index.js

```
> node dist/index.js
Hello
```

tsconfig.json

```
{
 "include": ["src/**/*.ts"],
 "compilerOptions": {
 "outDir": "dist",
 "target": "es6",
 "strict": true,
 "esModuleInterop": true,
 "module": "commonjs"
 }
}
```



# Serveur NodeJS avec Type Script et express

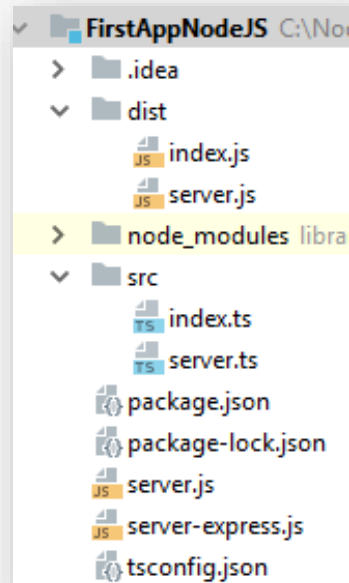
- Pour utiliser express avec type script, il faudrait en plus de l'installation de express, installer également le fichier de définition type script de express

```
> npm install --save @types/express
```

## index.ts

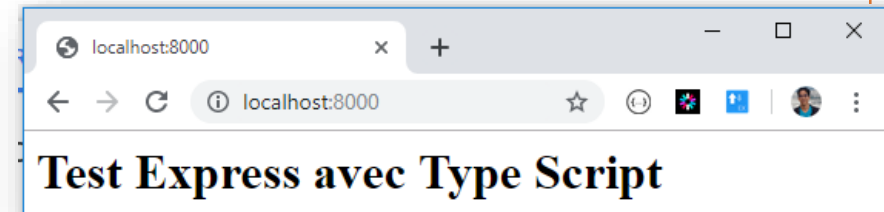
```
import Server from "./server";
const server=new Server(8000);
server.start();
```

```
> tsc
> node dist/index.js
server started ...
```



## server.ts

```
import express = require("express");
import {Request, Response} from "express";
export default class Server{
 constructor(private port:number){}
 public start():void{
 const app=express();
 app.get("/", (req:Request, res:Response)=>{
 res.send("<h1>Test Express avec Type
Script</h1>");
 });
 app.listen(this.port, ()=>{
 console.log("server started ...");
 });
 }
}
```





# Scripts en mode développement

## package.json

```
"scripts": {
 "start" : "npx tsc && node dist/index.js",
 "dev": "concurrently -n \"TS, Node\" \"npx tsc --watch\" \"nodemon dist/index.js\"",
},
```

> **npm run dev**

23:42:24 - Starting compilation in watch mode...

[TS]

[ Node] [nodemon] 1.19.1

[ Node] [nodemon] to restart at any time, enter `rs`

[ Node] [nodemon] watching: \*.\*

[ Node] [nodemon] starting `node dist/index.js`

[ Node] server started ...

# Tests Unitaire dans NodeJS avec jest

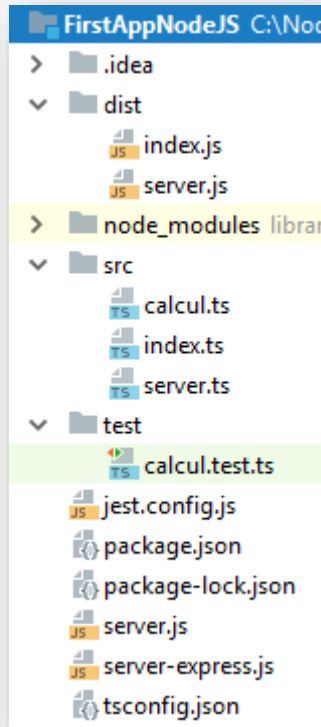
<https://jestjs.io/docs/en/getting-started.html>

- Jest est un framework de test JavaScript axé sur la simplicité.
- Il fonctionne avec des projets utilisant: Babel, TypeScript, Node, React, Angular, Vue et plus!

```
> npm install --save-dev jest
> npm install --save-dev @types/jest
> npm install --save-dev ts-jest
```

## jest.config.js

```
module.exports={
 transform:{
 "^.+\\.ts?$": "ts-jest"
 },
 testEnvironment: 'node',
 testMatch: ["**/test/*.test.ts"],
 moduleFileExtensions: ["js", "ts"]
}
```



## calcul.ts

```
export default class Calcul{
 public static somme(a:number,b:number){
 return a+b;
 }
}
```

## calcul.test.ts

```
import Calcul from "../src/calcul";
describe("Calcul", ()=>{
 it('should return 13', function () {
 let a:number=3;
 let b:number=10;
 let expected:number=13;
 expect(Calcul.somme(a,b)).toBe(expected);
 });
});
```

# Tests Unitaire dans NodeJS avec jest

```
C:\NodeJS\FirstAppNodeJS>npx jest
PASS test/calcul.test.ts
 Calcul
 ✓ should return 13 (3ms)

Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 0 total
Time: 2.869s
Ran all test suites.
```

## package.json

```
"scripts": {
 "test": "jest --verbose",
 "test-watch": "jest --verbose --watchAll",
 "start" : "npx tsc && node dist/index.js",
 "dev": "concurrently -n \"TS, Node\" \"npx tsc --watch\" \"nodemon dist/index.js\"",
},
```

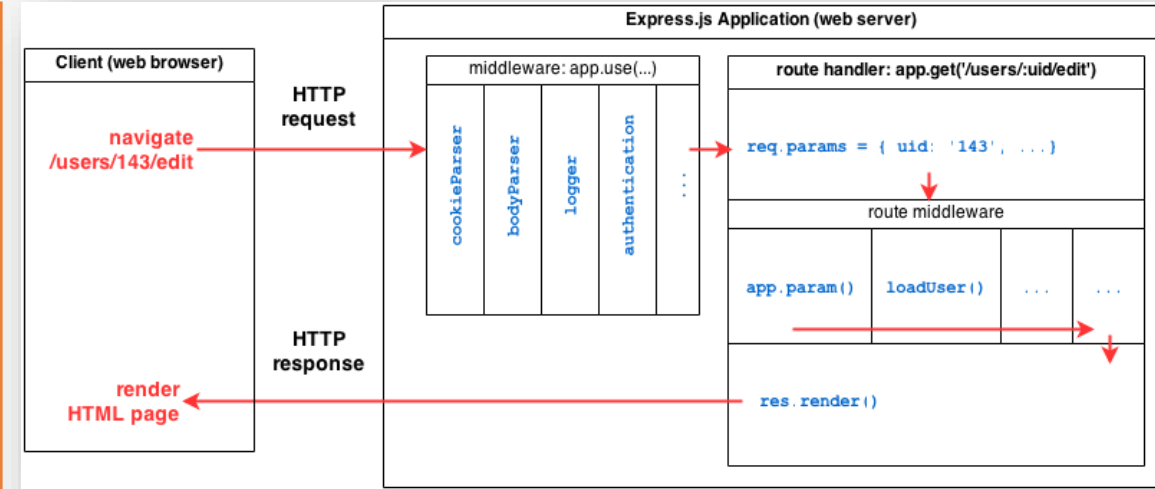
```
C:\NodeJS\FirstAppNodeJS>npm run test-watch
> FirstAppNodeJS@1.0.0 test-watch C:\NodeJS\FirstAppNodeJS
> jest --verbose --watchAll

RUNS test/calcul.test.ts
 Calcul
 ✓ should return 13 (3ms)

Test Suites: 1 passed, 1 total
Tests: 1 passed, 1 total
Snapshots: 0 total
Time: 1.743s, estimated 3s
Ran all test suites.
....
```

# Les middlewares de Express

- Express est un framework basé sur le concept de *middlewares*.
- Ce sont des petits morceaux d'application qui rendent chacun un service spécifique.
- Express est fourni avec une quinzaine de middlewares de base, et d'autres développeurs peuvent bien entendu en proposer d'autres via NPM.



- Les middlewares livrés avec Express fournissent chacun des micro-fonctionnalités. Il y a par exemple :
  - **body-parser** :
  - **compression** : permet la compression gzip de la page pour un envoi plus rapide au navigateur
  - **cookie-parser** : permet de manipuler les cookies
  - **cookie-session** : permet de gérer des informations de session (durant la visite d'un visiteur)
  - **serve-static** : permet de renvoyer des fichiers statiques contenus dans un dossier (images, fichiers à télécharger...)
  - **serve-favicon** : permet de renvoyer la favicon du site
  - **csurf** (csrf) : fournit une protection contre les failles CSRF
  - etc.

# Les middlewares de Express

- Ces middlewares sont interconnectés dans un pipe line et peuvent communiquer entre eux.
- Tous ces middlewares communiquent entre eux en se renvoyant jusqu'à 4 paramètres :
  - err : les erreurs
  - req : la requête du visiteur
  - res : la réponse à renvoyer (la page HTML et les informations d'en-tête)
  - next : un callback vers la prochaine fonction à appeler

Middleware 1 (err, req, res, next)



Middleware 1 (err, req, res, next)



Middleware 1 (err, req, res, next)

# Utiliser les middlewares dans Express

- Il suffit d'appeler la méthode `app.use()` pour utiliser un middleware.
- On peut enchaîner plusieurs `app.use("Mid1") .use("Mid2") .use("Mid3")`
- Par exemple :

```
import express, {Request, Response} from 'express';
import bodyParser from "body-parser";
import serveStatic from "serve-static";

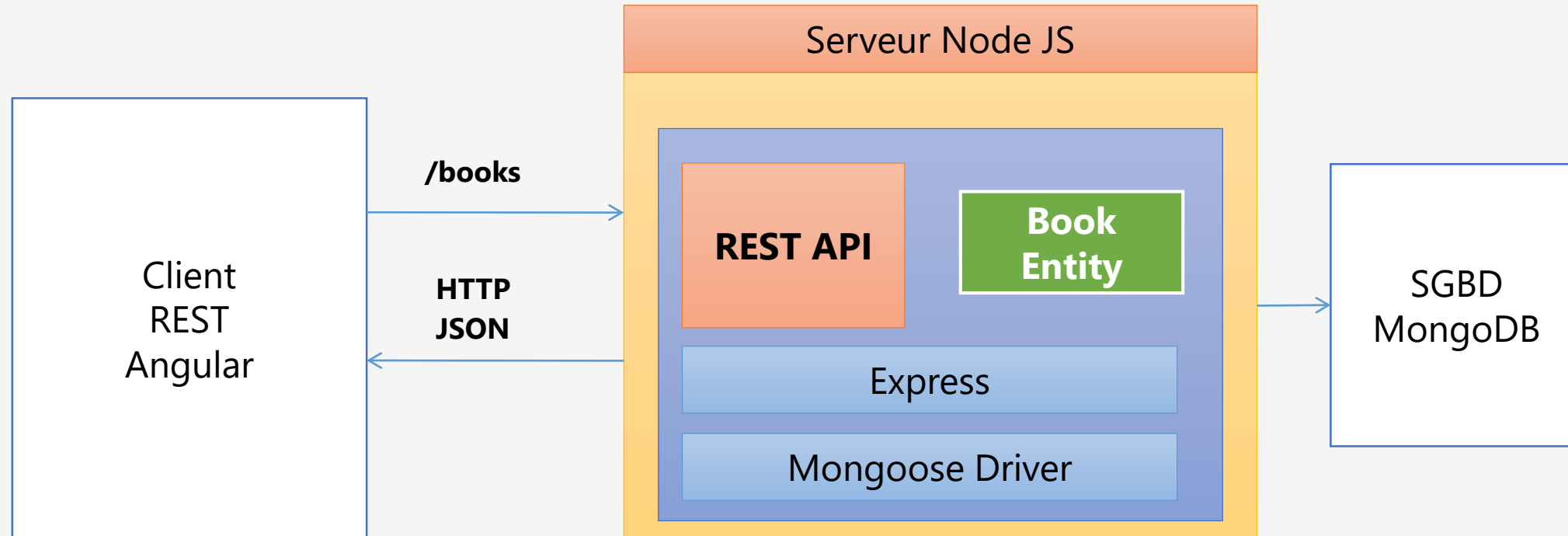
const app=express();

app.use(bodyParser.json()); // Retourne le middleware qui parse uniquement en Json

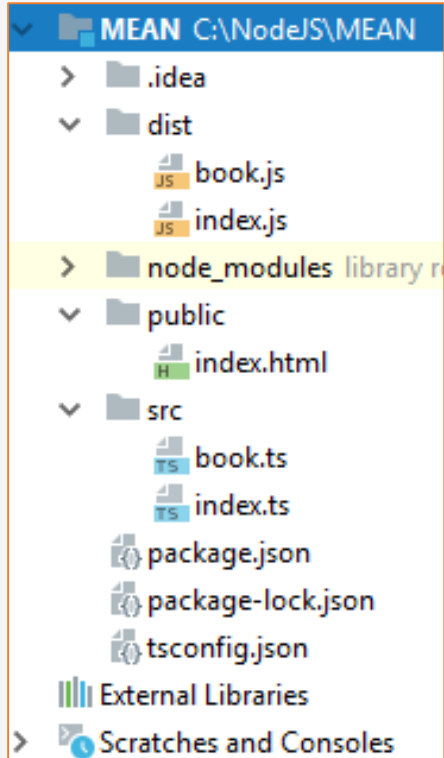
app.use(bodyParser());
app.use(serveStatic("public")); // Pour specifier le dossier des ressources statiques

app.listen(8700,()=>{
 console.log("Server Started on port %d",8700);
});
```

# API REST avec Node JS, Express, MongoDB



# Dépendances à installer



```
{
 "name": "MEAN", "version": "1.0.0", "description": "", "main": "index.js",
 "scripts": {
 "test": "jest --verbose",
 "test-watch": "jest --verbose --watchAll",
 "start": "npx tsc && node dist/index.js",
 "dev": "concurrently -n \"TS, Node\" \"npx tsc --watch\" \"nodemon dist/index.js\""
 },
 "keywords": [],
 "author": "", "license": "ISC",
 "dependencies": {
 "cors": "^2.8.5",
 "express": "^4.17.1",
 "mongoose": "^5.6.0",
 "mongoose-paginate": "^5.0.3"
 },
 "devDependencies": {
 "@types/express": "^4.17.0",
 "@types/mongoose": "^5.5.6",
 "@types/mongoose-paginate": "^5.0.6",
 "@types/cors": "^2.8.5",
 "concurrently": "^4.1.0",
 "nodemon": "^1.19.1",
 "typescript": "^3.5.2"
 }
}
```

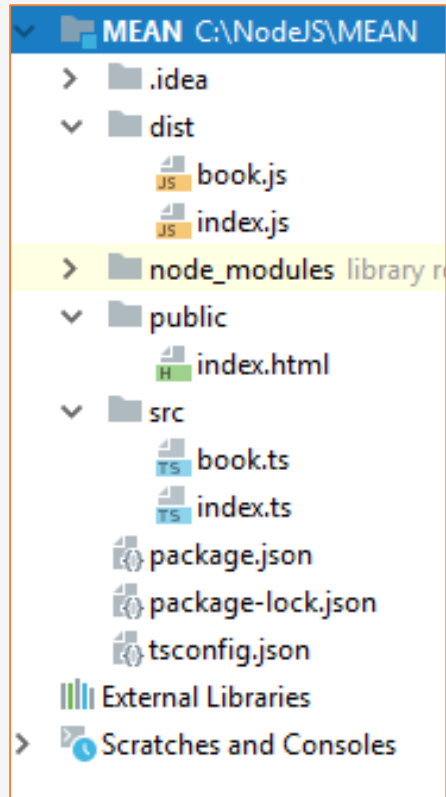
**package.json**

```
{
 "include": ["src/**/*.ts"],
 "compilerOptions": {
 "outDir": "dist",
 "target": "es6",
 "strict": true,
 "esModuleInterop": true,
 "module": "commonjs"
 }
}
```

**tsconfig.json**



# Création du Modèle



## book.ts

```
import mongoose from "mongoose";
import mongoosePaginate from "mongoose-paginate";

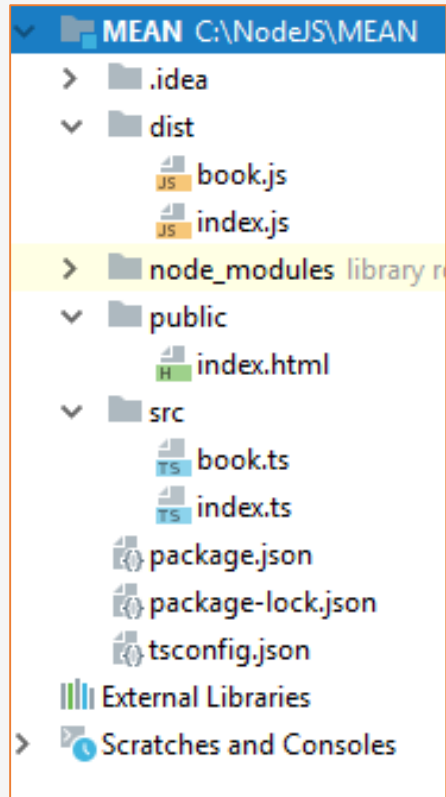
let bookSchema=new mongoose.Schema({
 title:{type:String,required:true },
 author:{type: String, required: true}
});

bookSchema.plugin(mongoosePaginate);

const Book=mongoose.model("Book",bookSchema);

export default Book;
```

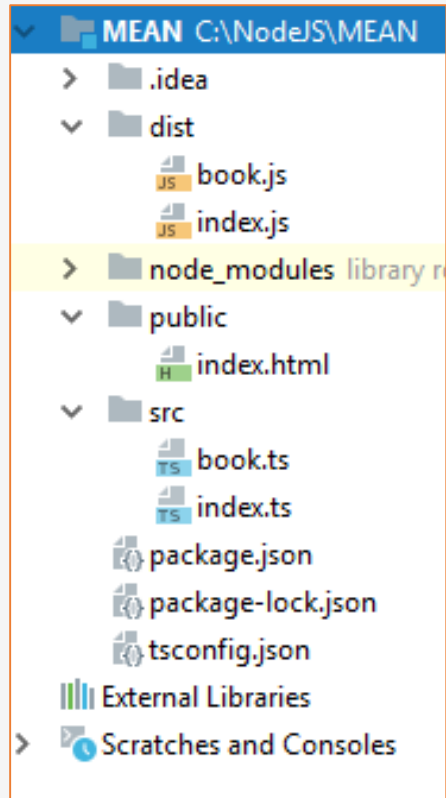
# Création du serveur Node JS



## index..ts

```
import express, {Request, Response} from 'express';
import Book from "./book";
import bodyParser from "body-parser";
import serveStatic from "serve-static";
import mongoose from "mongoose";
import cors from "cors";
/* Instancier Express */
const app=express();
/* Middleware bodyParser pour parser le corps des requêtes en Json*/
app.use(bodyParser.json());
/* Middleware pour configurer le dossier des ressources statique*/
app.use(serveStatic("public"));
/* Activer CORS*/
app.use(cors());
/* Connection à MongoDB*/
const uri:string="mongodb://localhost:27017/biblio";
mongoose.connect(uri, (err)=>{
 if(err){ console.log(err); }
 else{ console.log("Mongo db connection success"); }
});
```

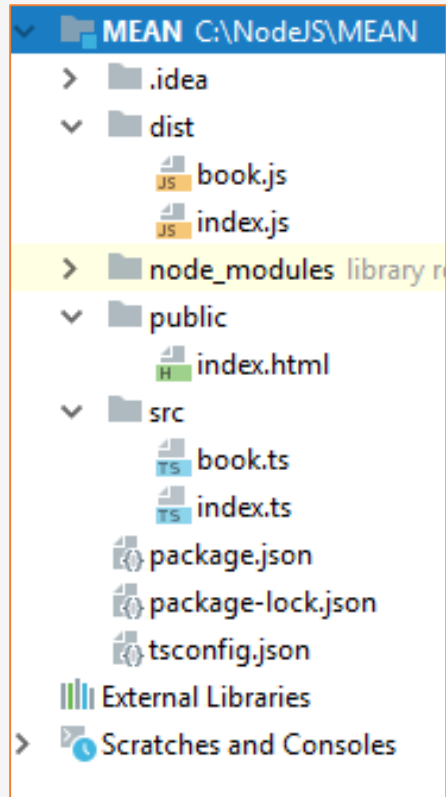
# Création du serveur Node JS (API Rest)



## index..ts

```
/* Requête HTTP GET http://localhost:8700/ */
app.get("/", (req: Request, resp: Response) => {
 resp.send("Hello world");
});
/* Requête HTTP GET http://localhost:8700/books */
app.get("/books", (req: Request, resp: Response) => {
 Book.find((err, books) => {
 if (err) { resp.status(500).send(err); }
 else { resp.send(books); }
 })
});
/* Requête HTTP GET http://localhost:8700/books/id */
app.get("/books/:id", (req: Request, resp: Response) => {
 Book.findById(req.params.id, (err, book) => {
 if (err) { resp.status(500).send(err); }
 else { resp.send(book); }
 });
});
```

# Création du serveur Node JS (API Rest)

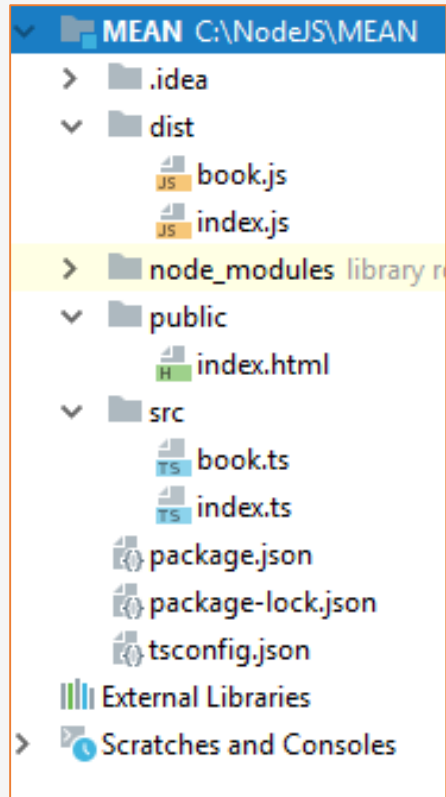


## index..ts

```
/* Requête HTTP POST http://localhost:8700/books */
app.post("/books", (req: Request, resp: Response) => {
 let book = new Book(req.body);
 book.save(err => {
 if (err) resp.status(500).send(err);
 else resp.send(book);
 })
});

/* Requête HTTP PUT http://localhost:8700/books/id */
app.put("/books/:id", (req: Request, resp: Response) => {
 Book.findByIdAndUpdate(req.params.id, req.body, (err, book) => {
 if (err) resp.status(500).send(err);
 else {
 resp.send("Successfully updated book");
 }
 })
});
```

# Création du serveur Node JS (API Rest)



## index..ts

```
/* Requête HTTP DELETE http://localhost:8700/books/id */
app.delete("/books/:id", (req: Request, resp: Response) => {
 Book.deleteOne({_id: req.params.id}, err => {
 if(err) resp.status(500).send(err);
 else resp.send("Successfully deleted Book");
 });
});

/* Démarrer Le serveur */
app.listen(8700, () => {
 console.log("Server Started on port %d", 8700);
});
```

# Création du serveur Node JS (API Rest)

## index..ts

```
/* Requête HTTP GET http://localhost:8700/pbooks?page=0&size=5 */
app.get("/pbooks", (req: Request, resp: Response) => {
 let p: number = parseInt(req.query.page || 1);
 let size: number = parseInt(req.query.size || 5);
 Book.paginate({}, { page: p, limit: size }, function(err, result) {
 if(err) resp.status(500).send(err);
 else resp.send(result);
 });
});
```

# Création du serveur Node JS (API Rest)

## index..ts

```
/* Requête HTTP GET http://localhost:8700/books-serach?kw=J&page=0&size=5 */
app.get("/books-serach", (req: Request, resp: Response) => {
 let p: number = parseInt(req.query.page || 1);
 let size: number = parseInt(req.query.size || 5);
 let keyword: string = req.query.kw || '';
 Book.paginate({ title: { $regex: ".*(?i)" + keyword + ".*" } }, { page: p, limit:
size }, function(err, result) {
 if(err) resp.status(500).send(err);
 else resp.send(result);
 });
});
```

# Test de l'API REST : Ajout avec POST

Method POST Request URL http://localhost:8700/books SEND

Parameters ^

Headers Body Variables

Body content type application/json Editor view Raw input

FORMAT JSON MINIFY JSON

```
{
 "title": "XML",
 "author": "Maya"
}
```

200 OK 130.83 ms

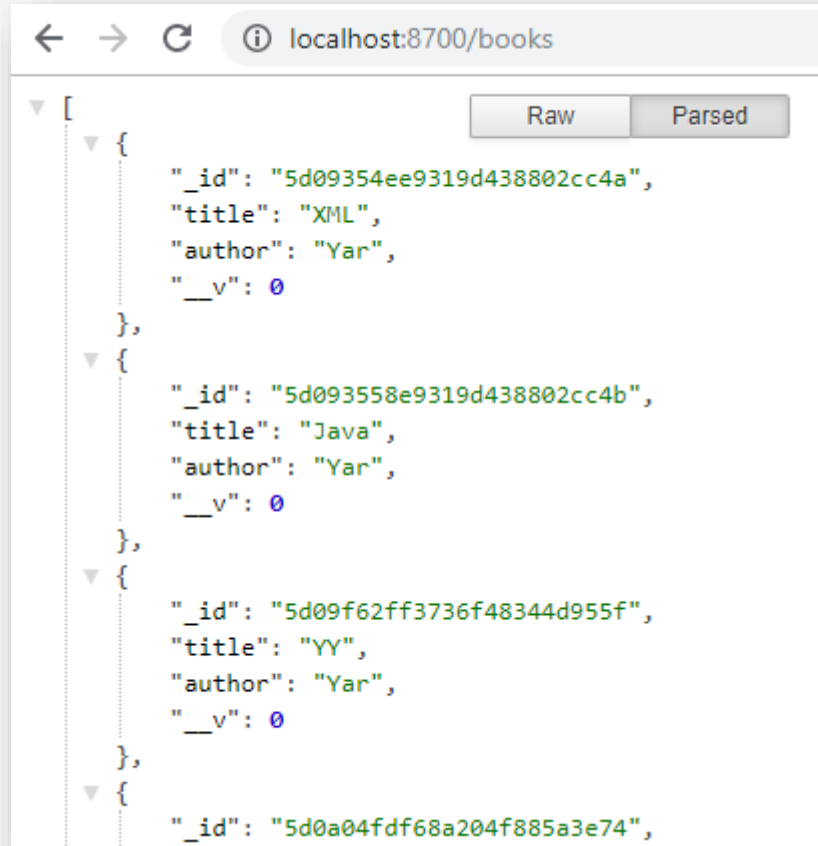
DETAILS v



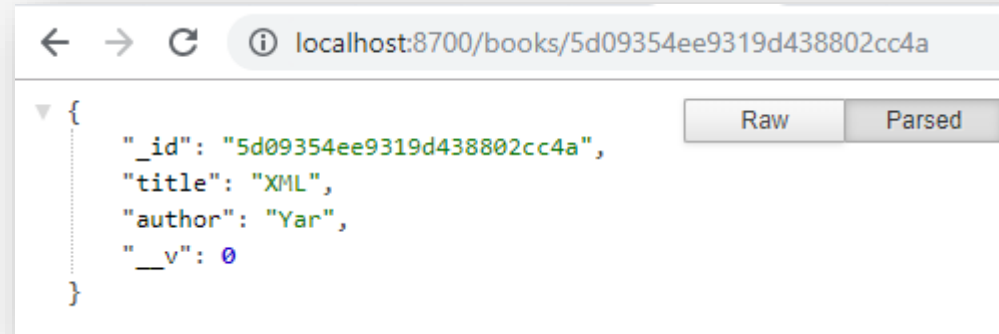
```
{
 "_id": "5d0a14ea3fb0043024e1a684",
 "title": "XML",
 "author": "Maya",
 "__v": 0
}
```



# Test de l'API REST : Consultation avec GET



```
[
 {
 "_id": "5d09354ee9319d438802cc4a",
 "title": "XML",
 "author": "Yar",
 "__v": 0
 },
 {
 "_id": "5d093558e9319d438802cc4b",
 "title": "Java",
 "author": "Yar",
 "__v": 0
 },
 {
 "_id": "5d09f62ff3736f48344d955f",
 "title": "YY",
 "author": "Yar",
 "__v": 0
 },
 {
 "_id": "5d0a04fdf68a204f885a3e74",
 "title": "ZZ",
 "author": "Yar",
 "__v": 0
 }
]
```



```
{
 "_id": "5d09354ee9319d438802cc4a",
 "title": "XML",
 "author": "Yar",
 "__v": 0
}
```

# Test de l'API REST : Mise à jour avec PUT

Method

PUT

Request URL

http://localhost:8700/books/5d09354ee9319d438802cc4a

SEND

Parameters

Headers

Body

Variables

Body content type

application/json

Editor view

Raw input

FORMAT JSON

MINIFY JSON

```
{
 "title": "JAVA",
 "author": "Maya"
}
```

200 OK

44.27 ms

DETAILS

Copy

Download

Code

Eye

Successfully updated book

# Test de l'API REST : Suppression avec DELETE

Method

DELETE

Request URL

http://localhost:8700/books/5d09354ee9319d438802cc4a



SEND


Parameters

Headers

Body

Variables

  Toggle source mode




 Insert headers set

Header name


Content-Type

Header value

application/json

ADD HEADER







Headers size: 30 bytes

200 OK

16.75 ms

DETAILS

Successfully deleted Book

# Test de l'API REST : Pagination

localhost:8700/pbooks?page=2&size=3

```
{
 "docs": [
 {
 "_id": "5d0a0502f68a204f885a3e75",
 "title": "YY",
 "author": "Yar",
 "__v": 0
 },
 {
 "_id": "5d0a0506f68a204f885a3e76",
 "title": "YY",
 "author": "Yar",
 "__v": 0
 },
 {
 "_id": "5d0a0b5b5a5319390c2d4d5a",
 "title": "YY",
 "author": "Yar",
 "__v": 0
 }
],
 "total": 7,
 "limit": 3,
 "page": 2,
 "pages": 3
}
```

localhost:8700/pbooks

```
{
 "docs": [
 {
 "_id": "5d093558e9319d438802cc4b",
 "title": "Java",
 "author": "Yar",
 "__v": 0
 },
 {
 "_id": "5d09f62ff3736f48344d955f",
 "title": "YY",
 "author": "Yar",
 "__v": 0
 },
 {
 "_id": "5d0a04fdf68a204f885a3e74",
 "title": "YY",
 "author": "Yar",
 "__v": 0
 },
 {
 "_id": "5d0a0502f68a204f885a3e75",
 "title": "YY",
 "author": "Yar",
 "__v": 0
 },
 {
 "_id": "5d0a0506f68a204f885a3e76",
 "title": "YY",
 "author": "Yar",
 "__v": 0
 }
],
 "total": 7,
 "limit": 5,
 "page": 1,
 "pages": 2
}
```

# Aperçu de la base de données MongoDB : Outils MongoDB Compass

MongoDB Compass Community - localhost:27017/biblio.books

Connect View Collection Help

My Cluster

localhost:27017 STANDALONE

17 DBS 25 COLLECTIONS

filter

- > Catal
- > admin
- ▼ biblio
  - books
  - > bourse\_db
  - > bourse\_db2
  - > catDB
  - > catalogogDB
  - > config
  - > db\_bourse
  - > db\_comp
  - > db\_comp2
  - > local
  - > movies\_db

biblio.books

Documents Aggregations Explain Plan

FILTER

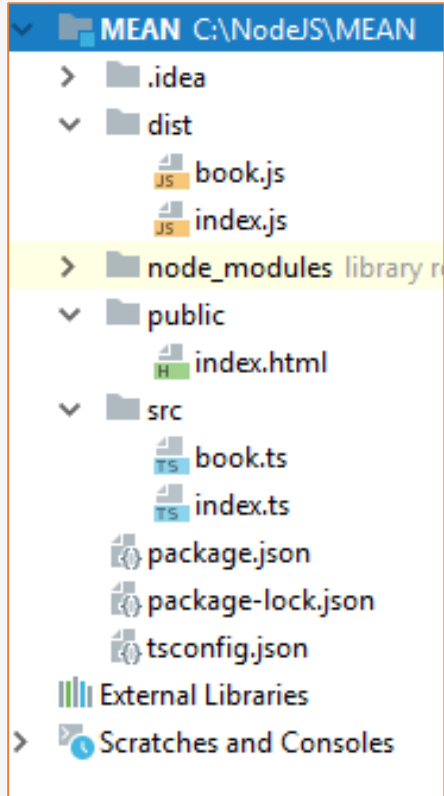
INSERT DOCUMENT VIEW LIST TABLE

```
{
 "_id": ObjectId("5d09354ee9319d438802cc4a"),
 "title": "XML",
 "author": "Yar",
 "__v": 0
}
```

```
{
 "_id": ObjectId("5d093558e9319d438802cc4b"),
 "title": "Java",
 "author": "Yar",
 "__v": 0
}
```

```
> {
 "_id": ObjectId("5d09f62ff3736f48344d955f"),
 "title": "YY",
 "author": "Yar",
 "__v": 0
}
```

# Mise à jour du Modèle



## book.ts

```
import mongoose, {Schema} from "mongoose";
import mongoosePaginate from "mongoose-paginate";
let bookSchema=new mongoose.Schema({
 title:{type:String,required:true },
 author:{type: String, required: true},
 price:{type:Number,required:false},
 available:{type:Boolean,required:true,default:false},
 publishingDate:{type:Date, required:true, default: new
Date()}}
});
bookSchema.plugin(mongoosePaginate);
const Book=mongoose.model("Book",bookSchema);
export default Book;
```

# Ajouter et Consulter un Livre

Method  
POST

Request URL  
http://localhost:8700/books

SEND

Parameters ^

Headers

Body

Variables

Body content type  
application/json

Editor view  
Raw input

FORMAT JSON

MINIFY JSON

```
{
 "title": "JAVA",
 "author": "Maya",
 "price": 89.90,
 "publishingDate": "2011-06-12"
}
```

200 OK 29.20 ms



```
{
 "available": false,
 "publishingDate": "2011-06-12T00:00:00.000Z",
 "_id": "5d0b62f5bb949139cc8e05d3",
 "title": "JAVA",
 "author": "Maya",
 "price": 89.9,
 "__v": 0
}
```

localhost:8700/books/5d0b62f5bb949139cc8e05d3

```
{
 "available": false,
 "publishingDate": "2011-06-12T00:00:00.000Z",
 "_id": "5d0b62f5bb949139cc8e05d3",
 "title": "JAVA",
 "author": "Maya",
 "price": 89.9,
 "__v": 0
}
```

# Chercher des livres

← → ↻ ⓘ localhost:8700/books-search?kw=J&page=1&size=3

```
{
 "docs": [
 {
 "available": false,
 "publishingDate": "2019-06-21T08:11:21.627Z",
 "_id": "5d093558e9319d438802cc4b",
 "title": "Java",
 "author": "Yar",
 "__v": 0
 },
 {
 "available": false,
 "publishingDate": "2019-06-21T08:11:21.627Z",
 "_id": "5d0b5f689840fa3d704b6c5e",
 "title": "JAVA",
 "author": "Maya",
 "__v": 0
 },
 {
 "available": false,
 "publishingDate": "2019-06-21T08:11:21.627Z",
 "_id": "5d0b5f789840fa3d704b6c5f",
 "title": "JAVA",
 "author": "Maya",
 "__v": 0
 }
],
 "total": 8,
 "limit": 3,
 "page": 1,
 "pages": 3
}
```



# Partie Front End avec Angular

Books List

Key word:

| Title  | Author | Price | Pub Date                 | Available |
|--------|--------|-------|--------------------------|-----------|
| XML    | Maya   | 800   | 2019-06-22T14:32:27.881Z | true      |
| JAVA   | Maya   | 600   | 2019-06-22T14:32:27.881Z | true      |
| JEE    | AZERTY | 120   | 2019-06-22T14:41:11.259Z | true      |
| Oracle | AZERTY | 11    | 2019-06-22T14:41:11.259Z | true      |
| ODBC   | AZERTY | 11    | 2019-06-22T14:41:11.259Z | true      |

0 1 2

# Partie Front End avec Angular

BiblioFrontWeb x +

localhost:4200/books-new

WebSiteName Home Books ▾ Page 2 Page 3

**Title**

JEE

**Author**

Taylor

**Publishing Date**

11/06/2019

**Price**

600

**Available**

☒

Save

BiblioFrontWeb x +

localhost:4200/books-new

WebSiteName Home Books ▾ Page 2 Page 3

New Book Added Successfully

**ID:** 5d0f636edd379031a88634a6

**Title:** JEE

**Author:** Taylor

**Pub Date:** 2019-06-11T00:00:00.000Z

**Price:** 600

**Available:** true

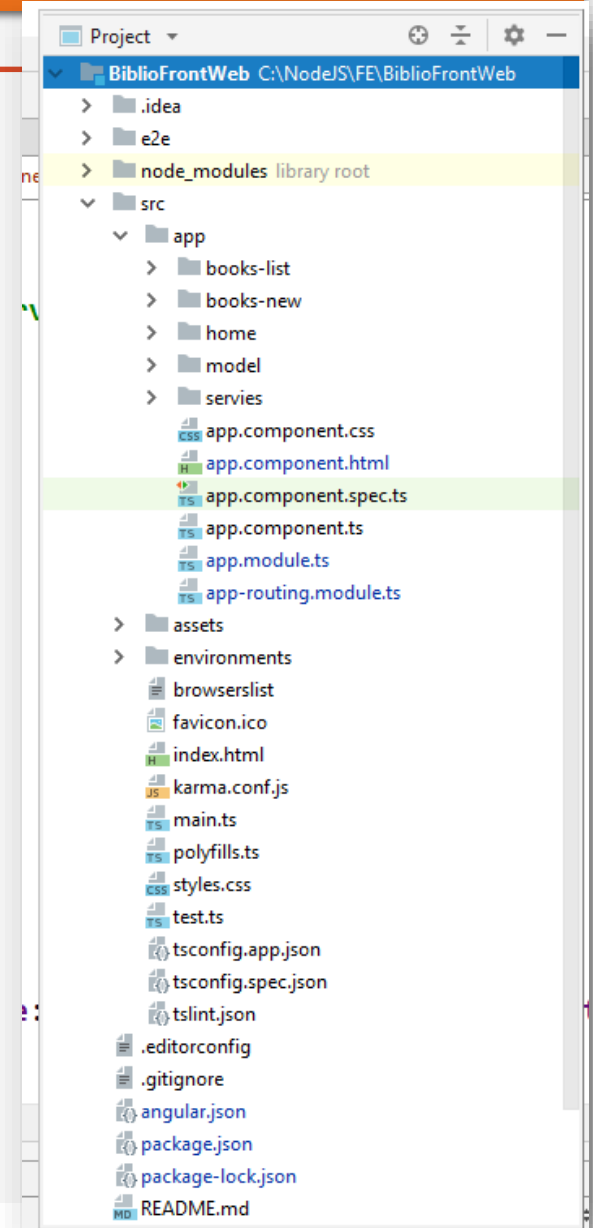
Close

# Structure du projet Angular

## package.json

```
{
 "name": "biblio-front-web",
 "version": "0.0.0",
 "scripts": {
 "ng": "ng", "start": "ng serve",
 "build": "ng build",
 "test": "ng test",
 "lint": "ng lint",
 "e2e": "ng e2e"
 },
 "private": true,
 "dependencies": {
 "@angular/animations": "~7.2.0",
 "@angular/common": "~7.2.0",
 "@angular/compiler": "~7.2.0",
 "@angular/core": "~7.2.0",
 "@angular/forms": "~7.2.0",
 "@angular/platform-browser": "~7.2.0",
 "@angular/platform-browser-dynamic": "~7.2.0",
 "@angular/router": "~7.2.0",
 "bootstrap": "^3.4.1",
 "core-js": "^2.5.4",
 "jquery": "^3.4.1",
 "rxjs": "~6.3.3",
 "tslib": "^1.9.0",
 "zone.js": "~0.8.26"
 },
```

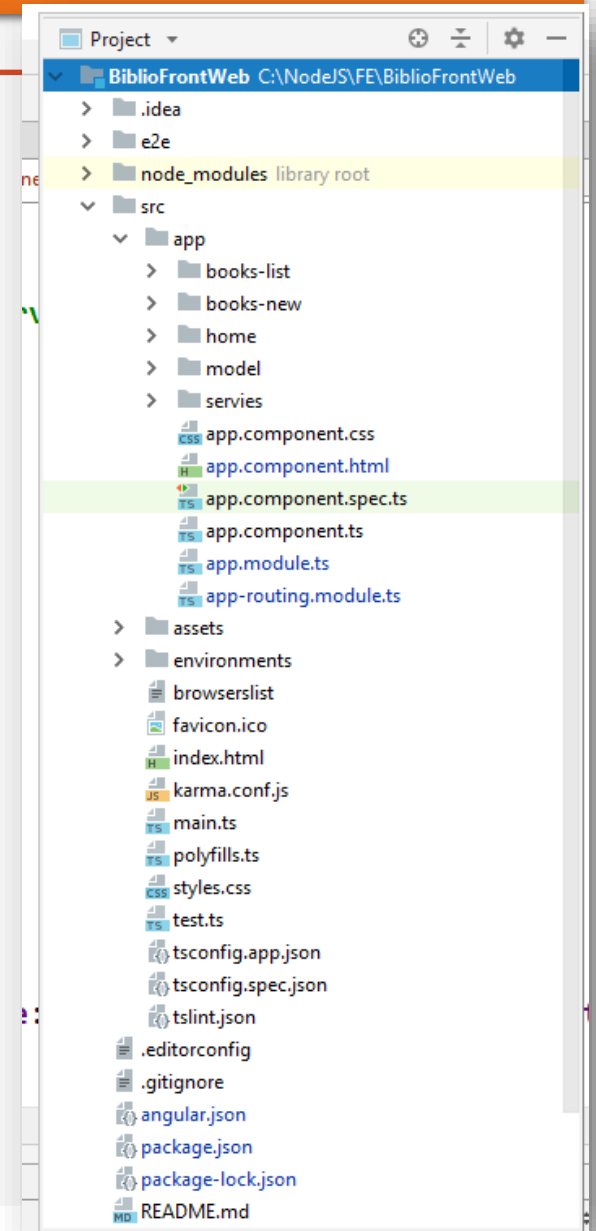
```
 "devDependencies": {
 "@angular-devkit/build-angular": "~0.13.0",
 "@angular/cli": "~7.3.1",
 "@angular/compiler-cli": "~7.2.0",
 "@angular/language-service": "~7.2.0",
 "@types/node": "~8.9.4",
 "@types/jasmine": "~2.8.8",
 "@types/jasminewd2": "~2.0.3",
 "codelyzer": "~4.5.0",
 "jasmine-core": "~2.99.1",
 "jasmine-spec-reporter": "~4.2.1",
 "karma": "~3.1.1",
 "karma-chrome-launcher": "~2.2.0",
 "karma-coverage-istanbul-reporter": "~2.0.1",
 "karma-jasmine": "~1.1.2",
 "karma-jasmine-html-reporter": "^0.2.2",
 "protractor": "~5.4.0",
 "ts-node": "~7.0.0",
 "tslint": "~5.11.0",
 "typescript": "~3.2.2"
 }
}
```



# Structure du projet Angular

## angular.json.json

```
"styles": [
 "src/styles.css",
 "node_modules/bootstrap/dist/css/bootstrap.min.css"
],
"scripts": [
 "node_modules/jquery/dist/jquery.min.js",
 "node_modules/bootstrap/dist/js/bootstrap.min.js"
],
```

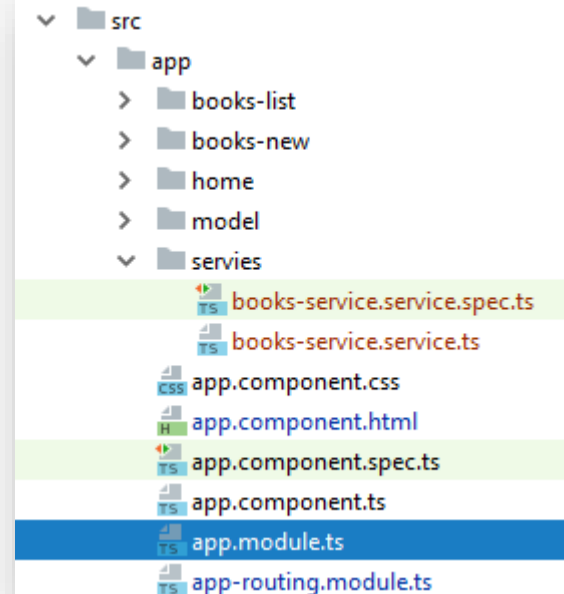


# Structure du projet Angular

## app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { BooksListComponent } from './books-list/books-list.component';
import { BooksNewComponent } from './books-new/books-new.component';
import { HomeComponent } from './home/home.component';
import { HttpClientModule } from '@angular/common/http';
import { FormsModule } from '@angular/forms';

@NgModule({
 declarations: [
 AppComponent, BooksListComponent, BooksNewComponent, HomeComponent
],
 imports: [
 BrowserModule, AppRoutingModule, HttpClientModule, FormsModule
],
 providers: [],
 bootstrap: [AppComponent]
})
export class AppModule { }
```



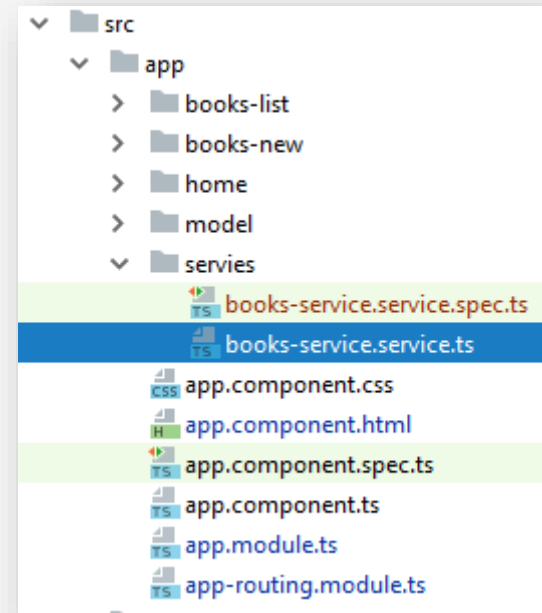
# Structure du projet Angular

## books.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Book, BookPage } from '../model/book.model';

@Injectable({
 providedIn: 'root'
})
export class BooksService {
 public host:string="http://localhost:8085";
 constructor(private httpClient:HttpClient) { }

 public searchBooks(keyword:string, page:number,
size:number):Observable<BookPage>{
 return this.httpClient.get<BookPage>(this.host+"/books-
search?kw="+keyword+"&page="+page+"&size="+size);
 }
 public saveBook(book:Book):Observable<Book>{
 return this.httpClient.post<Book>(this.host+"/books", book);
 }
}
```



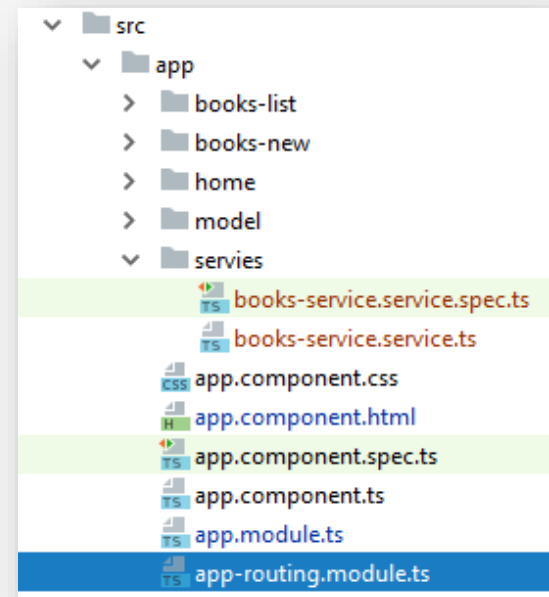
# Structure du projet Angular

## app-routing.module.ts.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { BooksListComponent } from '../books-list/books-list.component';
import { BooksNewComponent } from '../books-new/books-new.component';
import { HomeComponent } from '../home/home.component';

const routes: Routes = [
 {path:"books-list", component:BooksListComponent},
 {path:"books-new", component:BooksNewComponent},
 {path:"home", component:HomeComponent},
 {path:"", redirectTo:"/home", pathMatch:"full"},
];

@NgModule({
 imports: [RouterModule.forRoot(routes)],
 exports: [RouterModule]
})
export class AppRoutingModule { }
```



# Structure du projet Angular

## app.component.html

```
<nav class="navbar navbar-inverse">
 <div class="container-fluid">
 <div class="navbar-header">
 WebSiteName
 </div>
 <ul class="nav navbar-nav">
 <li class="active">Home
 <li class="dropdown">
 Books

 <ul class="dropdown-menu">
 Search
 New

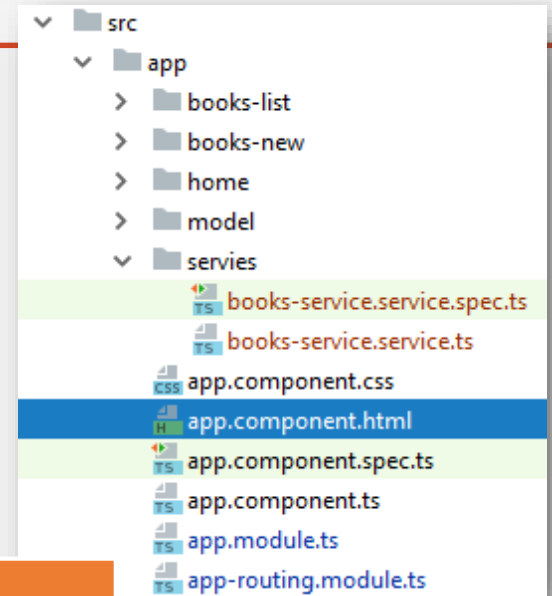
 Page 2
 Page 3

 </div>
</nav>
<router-outlet>
</router-outlet>
```

## app.component.ts

```
import { Component } from '@angular/core';

@Component({
 selector: 'app-root',
 templateUrl: './app.component.html',
 styleUrls: ['./app.component.css']
})
export class AppComponent {
 title = 'BiblioFrontWeb';
}
```





# Home Component

## home.component.html

```
<p>
 home works!
</p>
```

## app.component.ts

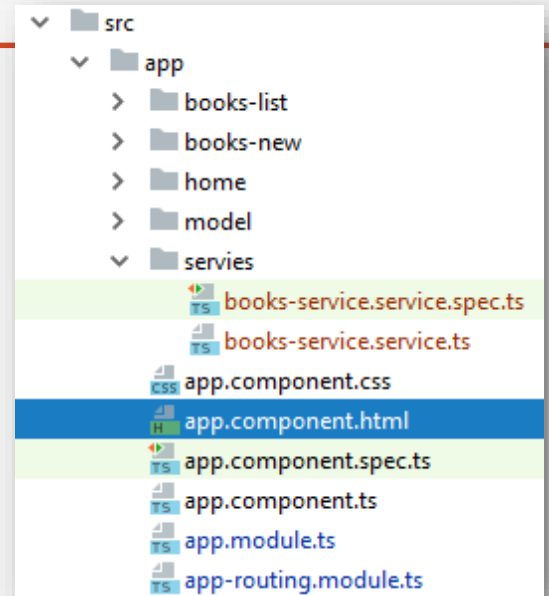
```
import { Component, OnInit } from
 '@angular/core';

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

 constructor() { }

 ngOnInit() {
 }

}
```

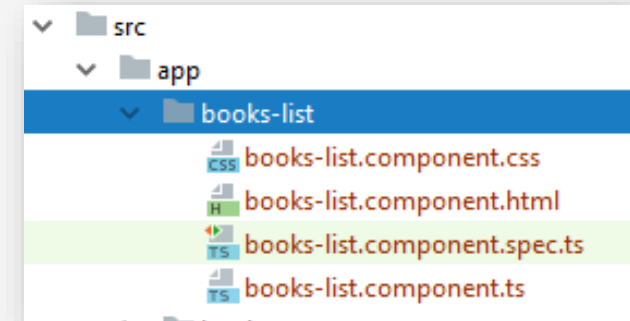


# BooksList Component

## books-list.component.html

```
<p></p>
<div class="container">
 <div class="panel panel-default">
 <div class="panel-heading">Books List</div>
 <div class="panel-body" *ngIf="books">
 <form #f="ngForm" (ngSubmit)="onSearch(f.value)">
 <div class="form-group">
 <label>Key word:</label>
 <input type="text" name="keyword" ngModel [(ngModel)]="keyword">
 <button type="submit">

 </button>
 </div>
 </form>
 </div>
 </div>
</div>
```

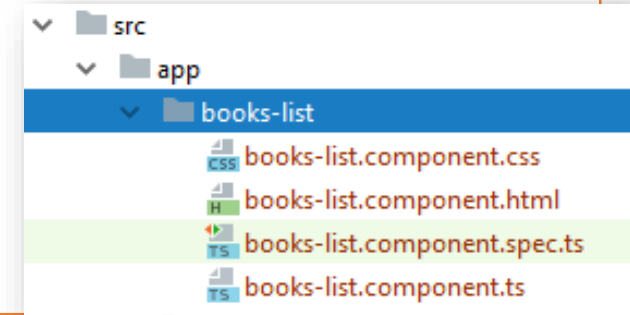


# BooksList Component

## books-list.component.html

```
<table class="table">
 <tr>
 <th>Title</th><th>Author</th><th>Price</th><th>Pub Date</th><th>Available</th>
 </tr>
 <tr *ngFor="let book of books.docs">
 <td>{{book.title}}</td> <td>{{book.author}}</td>
 <td>{{book.price}}</td> <td>{{book.publishingDate}}</td>
 <td>{{book.available}}</td><td>{{book.quantity}}</td>
 </tr>
</table>
<ul class="nav nav-pills">
 <li [ngClass]="((i+1)==currentPage)?'active':''" *ngFor="let page of pages; let i=index">
 {{i}}

</div>
</div>
</div>
```



# BooksList Component

## books-list.component.ts

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { BooksService } from '../services/books-service.service';
import { Book, BookPage } from '../model/book.model';
@Component({
 selector: 'app-books-list',
 templateUrl: './books-list.component.html',
 styleUrls: ['./books-list.component.css']
})
export class BooksListComponent implements OnInit {
 private books: BookPage;
 private keyword: string = "";
 private currentPage: number = 1;
 private pageSize: number = 5;
 private pages: Array<number>;

 constructor(private booksService: BooksService) { }

 ngOnInit() {
 this.onSearchBooks();
 }
}
```

```
private onSearchBooks() {
 this.booksService.searchBooks(this.keyword, this.currentPage, this.pageSize)
 .subscribe(data => {
 this.books = data;
 this.pages = new Array<number>(data.pages);
 }, err => {
 console.log(err);
 })
}

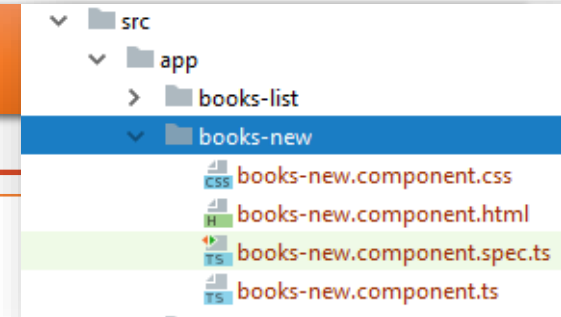
onPageBooks(i: number) {
 this.currentPage = i + 1;
 this.onSearchBooks();
}

onSearch(data) {
 console.log(data);
 this.keyword = data.keyword;
 this.onSearchBooks();
}
}
```

# BooksNew Component

books-new.component.html

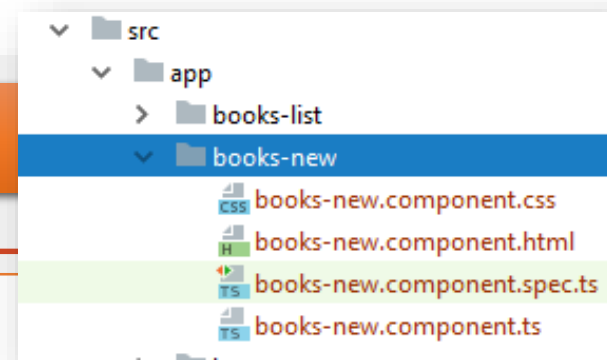
```
<p></p>
<div class="container">
 <div class="col-md-6 col-xs-12">
 <form #f="ngForm" (ngSubmit)="onSaveBook(f.value)" *ngIf="mode==0">
 <div class="form-group">
 <label class="control-label">Title</label>
 <input class="form-control" type="text" name="title" ngModel [(ngModel)]="book.title" required>
 </div>
 <div class="form-group">
 <label class="control-label">Author</label>
 <input class="form-control" type="text" name="author" ngModel [(ngModel)]="book.author" required>
 </div>
 <div class="form-group">
 <label class="control-label">Publishing Date</label>
 <input class="form-control" type="date" name="publishingDate" ngModel
[(ngModel)]="book.publishingDate" required>
 </div>
 <div class="form-group">
 <label class="control-label">Price</label>
 <input class="form-control" type="number" name="price" ngModel [(ngModel)]="book.price" required>
 </div>
 </form>
 </div>
</div>
```



# BooksNew Component

books-new.component.html

```
<div class="form-group">
 <label class="control-label">Available</label>
 <input class="checkbox" type="checkbox" name="available" ngModel [(ngModel)]="book.available"
required >
</div>
<button type="submit" class="btn btn-success">Save</button>
</form>
<div *ngIf="mode==1">
 <div class="panel panel-default">
 <div class="panel-heading">New Book Added Successfunly</div>
 <div class="panel-body">
 <div class="form-group"> <label>ID: {{book._id}}</label> </div>
 <div class="form-group"> <label>Title: {{book.title}} </label> </div>
 <div class="form-group"> <label>Author: {{book.author}}</label></div>
 <div class="form-group"> <label>Pub Date: {{book.publishingDate}}</label></div>
 <div class="form-group"> <label>Price: {{book.price}} </label></div>
 <div class="form-group"> <label>Available: {{book.available}}</label> </div>
 <button class="btn btn-success" (click)="onNewBook()">Close</button>
 </div>
 </div>
</div>
</div>
</div></div>
```



# BooksNew Component

## books-new.component.ts

```
import { Component, OnInit } from '@angular/core';
import { Book } from '../model/book.model';
import { BooksService } from '../services/books-service.service';

@Component({
 selector: 'app-books-new',
 templateUrl: './books-new.component.html',
 styleUrls: ['./books-new.component.css']
})
export class BooksNewComponent implements OnInit {
 public book: Book;
 public mode: number = 0;
 constructor(private booksService: BooksService) { }

 ngOnInit() {
 this.initBook();
 }
 private initBook() {
 this.book = { title: '', author: '', price: 0, publishingDate: new
Date(), available: true, quantity: 0 };
 }
}
```

```
onSaveBook(data: Book) {
 this.booksService.saveBook(data)
 .subscribe(res => {
 this.book = res;
 this.mode = 1;
 }, err => {
 console.log(err);
 })
}

onNewBook() {
 this.initBook();
 this.mode = 0;
}
}
```