

SHAKE THE FUTURE



Web Programming

ANGULAR CLI and NodeJS

JY Martin - JM Normand

Plan

1 Objectives

2 Angular

Main objective

In this practical work, we will work on Angular Framework.
We will also use NodeJS as a server.

The main objective is to use a Javascript based framework.
We could have use React or VueJS as well.

Why a JS framework?

- We use the same language in the FrontEnd and in the BackEnd
- It can be deployed on most environments with a single development

In fact, Angular is not really a JS Framework, but a Typescript framework. It is nearly the same because Javascript and Typescript are very close to each other.

Typescript is developed by Google.

Tools

To write this application you will need a text editor like Sublime Text, Notepad++, BBEdit, Atom, Visual Studio Code, and so on. You can also use Netbeans, Eclipse or IntelliJ if you want.

To display files, you will need a web browser with debugging tools.

You will need a JS server. We will use NodeJS.

And of course a Database Server. We will use PostgreSQL.

Tools: Node

Node is the server part, a JS server.

Alone it is not very user friendly, but with Express and a JS Framework, it can do many things.

Here is the location to download it.

<https://nodejs.org/en/download/>

Choose your installer, download it, launch it.

This will install node and npm, a package manager for node.

Tools: NPM

We have to ensure we have the latest version of npm.
Use your terminal / command tool

```
npm install -g npm@latest
```

Maybe :

- on macOS or linux you will have to be root.
sudo npm install ...
- on windows, you will have to launch the command window in administration mode.

Follow instructions.

Tools: Angular

Angular require Node to be installed. You will need at least version 10.9 of Node.

Why npm? Because we use it to download/install Angular.

```
npm install -g @angular/cli@latest
```


Tools: PostgreSQL

PostgreSQL should already be installed.
If not, have a look to prerequisites.

Plan

1 Objectives

2 Angular

Create new project

We use ng, an angular tool, to create our project. Go to your workspace and create the project.

```
ng new prwebANGULAR
```

- *reply "y" for angular routing** -take care, this is not the default value-
- use CSS for stylesheet format -this is the default setting-

That should create a folder prwebANGULAR.

Get inside: cd prwebANGULAR

Update project

Maybe some tools are deprecated. You should update them.
use npm for that

- npm install --save core-js
- npm install --save fsevents

You can also use this to get all outdated packages.

npm outdated

Check elements

Check Angular installed tools versions

```
ng -- version
```

```
Angular CLI
Angular CLI: 8.3.19
Node: 12.13.1
OS: darwin x64
Angular: 8.2.14
... animations, common, compiler, compiler-cli, core, forms
... language-service, platform-browser, platform-browser-dynamic
... router

Package                                  Version
-----
@angular-devkit/architect                0.803.19
@angular-devkit/build-angular            0.803.19
@angular-devkit/build-optimizer          0.803.19
@angular-devkit/build-webpack            0.803.19
@angular-devkit/core                     0.3.19
@angular-devkit/schematics               0.3.19
@angular/cli                             8.3.19
@ngtools/webpack                         8.3.19
@schematics/angular                     8.3.19
@schematics/update                       0.803.19
rxjs                                     6.4.0
typescript                               3.5.3
webpack                                  4.39.2
```

Check project

Now, we have to check it's ok.

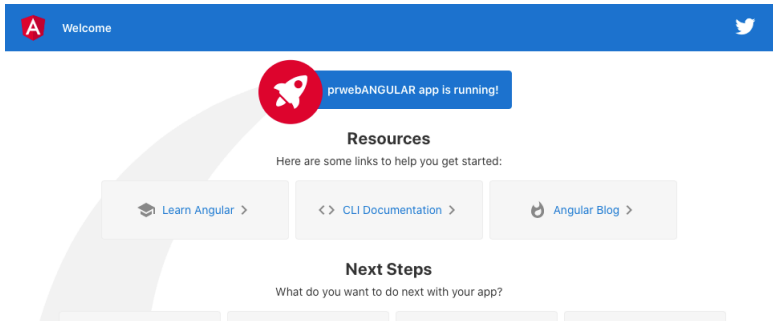
```
ng serve
```

First, Angular compile Typescript code. **ng build** would do the same. Then it copies files to folder dist. At last, it launches the server.

This should take some seconds to start.
Should be less than 1 minute.

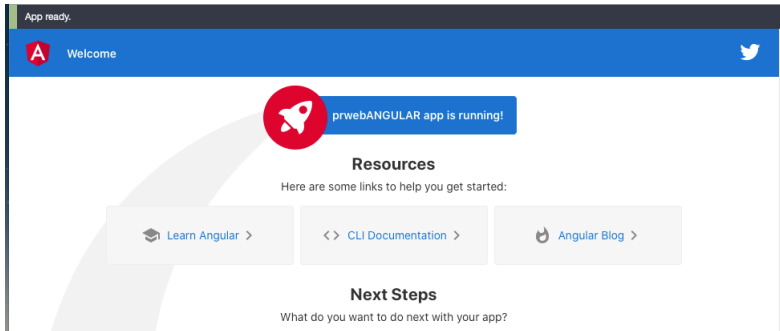
Check project

Now let's use your browser.
`http://localhost:4200`



Check project

You can also use the dev screen.
`http://localhost:4200/webpack-dev-server/`



The project

Here are the files in your project folder.



- file package.json contains installed modules list
- node_modules keep modules
- src contains your source files

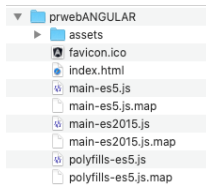
The project

Now have a look to the src folder.

- the **assets** folder will contain images, ...
- The **app** folder contains components. What is displayed. Each component is divided in several files:
 - a **css** file for the style
 - a **html** file for the content
 - a **ts** file for component elements
- the src folder contains already a component, with the 3 files.
- **environments** manages dev and prod environments. It contains ts files.

dist folder

Folder **dist** contains files to use by the server.
Angular copies required files in this folder before launching the server.



Components

Open the app.component.ts file in the src/app folder.

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

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

Have a look to @Component. Do you recognize some elements?

- Selector: have a look to the html file in src: index.html.
Do you see the tag "app-root"?
- templateUrl: have a look to app.component.html
- styleUrls: have a look to the css file

Creating a component

In Angular, components are the elements you use in your app.
In our case, we deal with auctions.

So, we will create a component "auctions".

Creating a component

Stop server using CTRL-C in your terminal.

Then create the component.

Note : g is for generate. You can use both of them.

ng g component auctions

Now, have a look to the src/app folder. You should have a new folder.

Our first page

So, let's try a first page with auctions.

From material, get several files:

- `auctions.html` contains elements for our first page.
- `main.css` is our stylesheet file

Our first page

Open file app.component.html
Remove lines inside big comments lines.

```
<!-- ***** -->
<!-- ***** The content below ***** -->
<!-- ***** is only a placeholder ***** -->
<!-- ***** and can be replaced. ***** -->
<!-- ***** -->
<!-- ***** Delete the template below ***** -->
<!-- ***** to get started with your project! ***** -->
<!-- ***** -->

<!-- ***** -->
<!-- ***** The content above ***** -->
<!-- ***** is only a placeholder ***** -->
<!-- ***** and can be replaced. ***** -->
<!-- ***** -->
<!-- ***** End of Placeholder ***** -->
<!-- ***** -->
```


Our first page

Open auctions.html in the materials. Copy elements inside the body tag.

Open file auctions.component.html in folder src/app/auctions. Copy lines from auction.html to this file. Copy only lines inside the tag body

```
1 <h1 id="homeTitle">List of items</h1>
2 <table>
3   <tr>
4     <th>Auction #</th>
5     <th>Auction Type</th>
6     <th>Seller</th>
7     <th>Description</th>
8     <th>Category</th>
9     <th></th>
10  </tr>
11    <tr>
12      <td>1</td>
13      <td>Computer</td>
14      <td>JY Martin</td>
15      <td>I sell my old computer
16      </td>
17      <td style="text-align:center">
18    </tr>
19    <tr id="addNew">
20      <td></td>
21      <td><input type="text" nam
22      <td><input type="text" nam
23      <td><input type="text" nam
24      <td></td>
25      <td style="text-align:center">
26    </tr>
27  </table>
```

Our first page

Launch server.

Connect to `http://localhost:4200`

Page is empty. Because we removed lines from `app.component.html`

Connect to `http://localhost:4200/auctions`

Page is empty too. Because we did not define the route to access this page.

Our first page

Have a look to file 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 { AuctionsComponent } from './auctions/auctions.component';

@NgModule({
  declarations: [
    AppComponent,
    AuctionsComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Do you see references to the auction component in the import instructions and in the declarations?

Our first page

Ok, let's define a route to our component.
Open file `app-routing.module.ts`

we will

- add component importation
- add route to auctions

Our first page

Have a look to file `app-routing.module.ts`

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

const routes: Routes = [];

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

This file manages routes the way you can access pages.
Routes are defined in the array **Routes**.

Our first page

Copy line about the auctions import in app.module.ts

```
import { AuctionsComponent } from './auctions/auctions.component';
```

Paste it in file app-routing.module.ts

```
import { NgModule } from '@angular/core';  
import { Routes, RouterModule } from '@angular/router';  
  
import { AuctionsComponent } from './auctions/auctions.component';
```

Our first page

Now let's add the route.

Create the route in the Routes array :

- define it as a new entry. Remember, it is a JSON file.
 - add path for the URL
 - and the component we refer to
 - ... you can also use data -we do not use it for this route-

```
const routes: Routes = [  
  { path: 'auctions', component: AuctionsComponent}  
];
```

Our first page

Now let's try.

If your server is stopped, launch it.



Connect to `http://localhost:4200`

Page is still empty. ok.

Connect to `http://localhost:4200/auctions`

Yess !!!!

List of items

Auction #	Auction Type	Seller	Description	Category
1	Computer	JY Martin	I sell my old computer, 3 years old	
	<input type="text"/>	<input type="text"/>	<input type="text"/>	

Our first page

What about images, styles, ...

First, CSS.

Remember ts files and the **styleUrls** line? You should find the same for the component

Open the file, copy text from main.css in the materials, to the angular css file.

And... have a look to the result. You don't have to restart the server, clear cache, ...

Our first page

List of items

Auction #	Auction Type	Seller	Description	Category	
1	Computer	JY Martin	I sell my old computer, 3 years old		
					

Our first page

Next, images.

They are assets too, so we have to put them in the right folder.

Copy icons folder from the materials in the **assets** folder in src.

Rename it as img.


Next, open auctions.component.html and change "img/..." to
"/assets/img/..."

And have a look to the result.

Maybe you will have to restart server if it is not taken into account.

Our first page

List of items

Auction #	Auction Type	Seller	Description	Category	
1	Computer	JY Martin	I sell my old computer, 3 years old		
					

Our first page

What about the default connection?

By default, we have an empty page when connecting to
`http://localhost:4200`

Couldn't we redirect this page to the list of items?

Open the routing file. Add this line in the Routes :

```
{ path: '', redirectTo: 'auctions', pathMatch: 'full'}
```

Take care, path " is 2 single quotes.

This route will redirect to route "auctions".

Remember, Routes is an array. Do not forget comma between routes definition.

Client-Server interaction

Now, we would like the server and the client to interact.
This is done with **services**.

Services are special components.
Let's create a service. So, from the terminal

```
ng g service auction
```

This should create 2 files in src/app

Client-Server interaction

Have a look to auction.service.ts

```
import { Injectable } from '@angular/core';  
  
@Injectable({  
  providedIn: 'root'  
})  
export class AuctionService {  
  
  constructor() { }  
}
```

This defines a new type of component : Injectable.
It also create class AuctionService.

Client-Server interaction

Have a look to `auction.service.spec.ts`

```
import { TestBed } from '@angular/core/testing';  
  
import { AuctionService } from './auction.service';  
  
describe('AuctionService', () => {  
  beforeEach(() => TestBed.configureTestingModule({}));  
  
  it('should be created', () => {  
    const service: AuctionService = TestBed.get(AuctionService);  
    expect(service).toBeTruthy();  
  });  
});
```

Have a look to the way we import `AuctionService`.

Client-Server interaction

Go back to auction.service.ts
Let's create a method inside.
This method create a line for the table.

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

@Injectable({
  providedIn: 'root'
})
export class AuctionService {

  constructor() { }

  loadAll() {
    var jsonString = '{'
      + '"id": 1, '
      + '"title": "Computer", '
      + '"author": "JM Normand", '
      + '"body": "Looking for new modeling tool."'
      + '}';
    var returned = JSON.parse(jsonString);
    return returned;
  }
}
```

We create a string with JSON elements, and parse it to create a JSON object.

Client-Server interaction

Ok, now let's call it from the client, the auctions.

First, add the service call in `auctions.component.ts`

- import the service. Remember, auction is in a folder, so we have to load it from parent folder.
- load the auctions from the service.
 - add a attribute to save them locally, we declare it as an object.
 - declare the service. We add it in the constructor. The name of the variable will be `_auctions`, but you can take the one you want.
 - call the service in the `init` method, and save result in the variable.

Client-Server interaction

```
import { Component, OnInit } from '@angular/core';
import { AuctionService } from '../auction.service';

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

  auctions: Object;

  constructor( private _auctions: AuctionService ) {}

  ngOnInit() {
    this.auctions = this._auctions.loadAll();
  }
}
```

Client-Server interaction

Next, add it to the html file.
To use a variable we use this:

```
{{ myVariable }}
```

Of course, if it is an object, that will lead to this:

```
{{ myVariable.myAttribute }}
```

Client-Server interaction



Our variable is auctions, the one we defined in the ts file.
Do you remember the object we created in the service, the name of the attributes?

```
<tr>
  <th>Auction #</th>
  <th>Auction Type</th>
  <th>Seller</th>
  <th>Description</th>
  <th>Category</th>
  <th></th>
</tr>
<tr>
  <td>{{ auctions.id }}</td>
  <td>{{ auctions.title }}</td>
  <td>{{ auctions.author }}</td>
  <td>{{ auctions.body }}</td>
  <td></td>
  <td style="text-align:center"><button></button></td>
</tr>
```

Client-Server interaction

Let's see the result:

List of items

Auction #	Auction Type	Seller	Description	Category	
1	Computer	JM Normand	Looking for new modeling tool.		
					

Client-Server interaction

Ok. Let's try with more auctions.



First we change the generation in the service:

```
var jsonString = '[  
  {  
    "id": 1, '  
    "title": "Computer", '  
    "author": "JM Normand", '  
    "body": "Looking for new modeling tool."  
  },  
  {  
    "id": 2, '  
    "title": "Anime", '  
    "author": "JY Martin", '  
    "body": "Who has the full Goblin slayer anime?"  
  }  
' ]';
```

Client-Server interaction

Here is the result.

List of items

Auction #	Auction Type	Seller	Description	Category	
					
					

But... oh, this is an array. We need a loop to get the auctions.

Client-Server interaction




Here is the way we ask angular to loop on a variable.
We add an **ng** loop with attribute *ngFor. Then we give the iteration variable, and the array we loop on.

```
<tr *ngFor="let auction of auctions ">
  <td>{{ auction.id }}</td>
  <td>{{ auction.title }}</td>
  <td>{{ auction.author }}</td>
  <td>{{ auction.body }}</td>
  <td></td>
  <td style="text-align:center"><button></button></td>
</tr>
```

Client-Server interaction

Here is the result.

List of items

Auction #	Auction Type	Seller	Description	Category	
1	Computer	JM Normand	Looking for new modeling tool.		
2	Anime	JY Martin	Who has the full Goblin slayer anime?		
					

Client-Server interaction

Yes but... this is not a real client-server interaction. This is only a service.

That means our code - components, html, css, ...- are sent to the browser.

But we want to connect to a database, and the browser will not be able to. We need a backend server that will connect to the database.

Client-Server interaction

Let's create a server.

Go to the root of your project. Create folder "server".

In a new terminal, go inside the server folder. Create server data:

```
npm init
```

Answer questions.

Your answers will be in package.json

Client-Server interaction

We will need some tools too.

```
npm install express --save  
npm install pg --save
```

- express manages our application.
- pg allows us to interact with the PostgreSQL database.

Client-Server interaction

In your server folder, create a file server.js with this content.

```
const express = require('express');  
const app = express();  
  
app.listen(8000, () => {  
  console.log('Server started!')  
});
```

First lines are initialisations.

Last bloc, listen lines, explains our server is listening to port 8000.

Client-Server interaction

Ok, now, we need a method that should reply to a basic request. We will use the database a bit later. Add these lines to the server file, just before the “listen” lines.

```
app.get('/', function(req, res) {  
  var jsonString = '['  
    + '{'  
      + '"id": 1, '  
      + '"title": "Computer", '  
      + '"author" : "JM Normand", '  
      + '"body" : "Looking for new modeling tool."'  
    + '},'  
    + '{'  
      + '"id": 2, '  
      + '"title": "Anime", '  
      + '"author" : "JY Martin", '  
      + '"body" : "Who has the full Goblin slayer anime?"'  
    + '}'  
    + ']'  
  res.setHeader('Content-Type', 'application/json');  
  res.send(jsonString);  
});
```

app.get means your application will reply to GET requests. URL we reply to is the first parameter: “/”

Client-Server interaction

Open a new terminal, yes another one. Go to the server location.
Launch node server:

```
node server.js
```

or better,

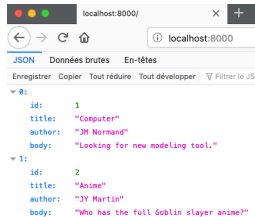
```
npm start
```


Client-Server interaction

And, in your browser, use URL `http://localhost:8000`
That should give the one of following result:



```
[{"id": 1, "title": "Computer", "author": "JM Normand", "body": "Looking for new modeling tool."}, {"id": 2, "title": "Anime", "author": "JY Martin", "body": "Who has the full Goblin slayer anime?"}]
```



localhost:8000/

localhost:8000

JSON Données brutes En-têtes

Enregistrer Copier Tout réduire Tout développer Filtre le JS

0:

- id: 1
- title: "Computer"
- author: "JM Normand"
- body: "Looking for new modeling tool."

1:

- id: 2
- title: "Anime"
- author: "JY Martin"
- body: "Who has the full Goblin slayer anime?"

Client-Server interaction

Now, we link the Angular service and the node server.

Service is on the browser.

Server is... on the server.

So, the browser should use a http request to connect to the server to get data.

For that we use an Angular module : HttpClient.

Client-Server interaction

So, first, add HttpClient to the files.

- in app.module.ts
 - import HttpClientModule from '@angular/common/http'
 - add HttpClientModule in the imports
- in auction.service.ts
 - import HttpClient from '@angular/common/http';
 - change AuctionService by the one next slide

Client-Server interaction

Then we link the service to the server.

First, we define a string that is the link to the node server.

Then we add an HttpClient as a parameter to the constructor to be able to use the link.

At last, we call it.

```
export class AuctionService {  
  private _serverURL = 'http://localhost:8000';  
  constructor(private _httpClient: HttpClient) { }  
  loadAll() {  
    return this._httpClient.get(this._serverURL);  
  }  
}
```

Client-Server interaction

Be sure your node server is running.
Run your angular server.

And... It doesn't work. Why?

- Angular server is on port 4200
- Node server is on port 8000

A browser, by default is not allowed to connect to a different server or on a different port.

We have to fix that.

Client-Server interaction

Come back to the server.js file and add following lines, just after const declarations.

```
app.use(function(req, res, next) {  
  res.header("Access-Control-Allow-Origin", "*");  
  res.header("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");  
  next();  
});
```

These lines says it can reply to request from any server.

Client-Server interaction

Let's try again.

And... It doesn't work either. Why?

- calls with HttpClient return Observables. Observables are Object not fully evaluated.
- method LoadAll gets the result as a JSON object.

So the returned object from the service -Observable- and the object used in the component -JSON- do not match.
We will fix that too.

Client-Server interaction

Observable? What is that?

This kind of object was created to get objects that could change. it is a "Reactive" object.

The consequence is that you cannot get its value like you would do for a variable. Affectation to this object is asynchronous. One more: is a lazy evaluation. You will have to tell when you need the value.

So, we have to wait the object gets its value.

For that, we will use a method: **subscribe** that forces variable evaluation.

Client-Server interaction

Change the auctions.component.ts file.




- We load the result, the Observable
- we force its evaluation with subscribe
- and we save the result in the local variable.

```
ngOnInit() {  
  this._auctions.loadAll().subscribe(auctions => this.auctions = auctions);  
}
```

Client-Server interaction

One more try... it... works!!! yes!!!

List of items

Auction #	Auction Type	Seller	Description	Category	
1	Computer	JM Normand	Looking for new modeling tool.		
2	Anime	JY Martin	Who has the full Goblin slayer anime?		
					

Database interaction

We should not use static data. Our data are in a database. So we need to interact with the database to get data. On the server.js file.

- First add pg declaration at the beginning of the file, with other requirements.

```
const express = require('express');  
const pg = require("pg");  
const app = express();
```

- Then add connection string. It's a string, not the real connection.
Feel free to change login, password, and the database name according to yours.

```
var conString = 'postgres://login:password@localhost:5432/database';
```

Database interaction

Next, getting informations from database.

From pg, we have to:

- Create a connection to the database
- If it's ok, send on request on table item
- at last build a JSON object and send it back.

The functions we use are asynchronous functions. So, never call them as a list of instructions. If the first action is not ended, launching the second one will fail.

Have a look next slide to see how you can process it.

Database interaction

Here is the script of the app.get function you have to replace:

```
app.get('/', function(req, res) {
  var client = new pg.Client(conString);
  client.connect(function(err) {
    if (err) {
      // Cannot connect
      console.error('could not connect to postgres', err);
      res.status(500).end('Database connection error!');
    } else {
      client.query('SELECT * FROM item ORDER BY id', function(err, result) {
        if (err) {
          // Request fails
          console.error('bad request', err);
          res.status(500).end('Bad request error!');
        } else {
          // Build result
          var items = [];
          for (var ind in result.rows) {
            items.push(result.rows[ind]);
          }
          jsonString = JSON.stringify(items);
          res.setHeader('Content-Type', 'application/json');
          res.send(jsonString);
        }
        client.end();
      });
    }
  });
});
```

Database interaction

Check it works.

Relaunch node server.

Use your browser and call <http://localhost:8000/>









You should have something like this.

```
[{"id":2,"title":"Anime","author":"  
{"id":3,"title":"Elenium","author":  
{"id":4,"title":"Kinect","author":  
{"id":5,"title":"Kikou","author":  
{"id":6,"title":"Mangas","author":  
{"id":7,"title":"Anime","author":
```

Database interaction

Update your browser.

List of items

Auction #	Auction Type	Seller	Description	Category	
2	Anime	JY Martin	Looking for the end of the Fairy Tail manga (> 126)		
3	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings		
4	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer		
5	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?		
6	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?		
7	Anime	JY Martin	Looking for full version of Daimachi		
8	Anime	JY Martin	Selling LODOSS war		
					

Database interaction

As node will only serve data, it should reply only to predefined routes.

So, change the URL of the app.get method in server.js by **/listAuctions**.

Also change it into auction.service.ts

```
loadAll() {  
    return this._httpClient.get(this._serverURL+"/listAuctions");  
}
```

Check its still ok, on port 8000 and on port 4200.

Database interaction








Now let's add categories. Change SQL request in the server by this one:

```
SELECT item.*, name  
FROM item  
JOIN Category ON (item.category_id=category.id)  
ORDER BY item.id
```

Now, you've get name, the category's name, so you can use it in the html display.

Database interaction

List of items

Auction #	Auction Type	Seller	Description	Category	
2	Anime	JY Martin	Looking for the end of the Fairy Tail manga (> 126)	books	
3	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	books	
4	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	electronics	
5	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?	pets	
6	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	books	
8	Anime	JY Martin	Selling LODOSS war	electronics	
					









Something missing? Maybe some lines.

Database interaction

What if the auction has no category?

We should use "LEFT OUTER JOIN" instead of the "INNER JOIN".

List of items

Auction #	Auction Type	Seller	Description	Category	
2	Anime	JY Martin	Looking for the end of the Fairy Tail manga (> 126)	books	
3	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	books	
4	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	electronics	
5	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?	pets	
6	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	books	
7	Anime	JY Martin	Looking for full version of Daimachi		
8	Anime	JY Martin	Selling LODOSS war	electronics	
					

Database interaction

Let's add the category list when adding a new auction.
We need to

- Add a method `listCategories` in the node server
- Add a service to get categories
- Add a select tag in the HTML page
- Get categories when the application starts

Database interaction

First, the node server.

Add a method with route `listCategories` in `server.js`
Use the right SQL request to get categories.

Relaunch node server.
Check it works on port 8000.

Database interaction

Next, the service.

- Create a service category in the application.
- Add http import, as we did for auctions.
- Create method loadAll like we did for auctions.
Launch a request to the server to get listCategories.

Database interaction

Next, HTML page.

In the `auctions.component.html` file.

Display a list selector in the right cell.

We loop on the options to parse categories and add them to the list.

```
<td><select name="category" id="category">
  <option value="-1">--</option>
  <option *ngFor="let category of categories" value="{{ category.id }}">{{ category.name }}</option>
</select>
</td>
```

Database interaction









Last, initialise category list when page starts.
In the `auctions.component.ts` file.

- Add Category Service in the imports.
- Add an object "categories" in the class attributes, like "auctions".
- Add a CategoryService parameter to the constructor.
- And call method `loadAll` from the category service in the `ngOnInit` method.

Database interaction

Let's try all.

List of items

Auction #	Auction Type	Seller	Description	Category	
2	Anime	JY Martin	Looking for the end of the Fairy Tail manga (> 126)	books	
3	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	books	
4	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	electronics	
5	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?	pets	
6	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	books	
7	Anime	JY Martin	Looking for full version of Daimachi		
8	Anime	JY Martin	Selling LODOSS war	electronics	
				-	

A bit of factorisation

Did you notice that `listAuctions` and `listCategories` are nearly the same?

Maybe we could write a single function with the SQL request as a parameter...

Create a function in `server.js`
Something like this:

```
function listAll(req, res, tableReq) {  
  ...  
  client.query(tableReq, function(err, result) {  
    ...  
  }  
  
  app.get('/listCategories', function(req, res) {  
    var tableReq = 'SELECT * FROM category ORDER BY id';  
    listAll(req, res, tableReq);  
  });  
}
```

Call the function for `listAuctions` and `listCategories`.

A bit of factorisation

Our function is usefull but what if there are parameters in the SQL request?

Let's add a parameter to use values. The query function uses an array to manage parameters.

Our SQL request has no parameter, so we use an empty array.

```
function listAll(req, res, tableReq, values) {  
  ...  
  client.query(tableReq, values, function(err, result) {  
    ...  
  }  
}  
  
app.get('/listCategories', function(req, res) {  
  var tableReq = 'SELECT * FROM category ORDER BY id';  
  var values = [];  
  listAll(req, res, tableReq, values);  
});
```

A bit of factorisation

Now let's try to use parameters in our SQL requests.
Maybe we could write a method that gets a category from its ID.

- our JS function must have a parameter. Using REST, this should look like this.

```
app.get('/getCategory/:id', function(req, res) { ... });
```

- SQL query may include references to parameters with \$1, \$2, ...

```
var tableReq = 'SELECT * FROM Category WHERE id=$1';  
var values = [id];
```

Relaunch node server and test it
<http://127.0.0.1:8000/getCategory/1>

Adding auctions

Ok, now we focus on adding an auction.

First we have to tell the green "+" button that we add the auction.
Let's modify button "+" in the file auction.component.html

```
<td style="text-align:center">  
  <button (click)="onSave($event)">  
      
  </button>  
</td>
```

This tells button + that, when a **click** event occurs on it, it has to execute function onSave with the event as a parameter.

Adding auctions

Ok, now the function `onSave`. It is located in the `ts` file, like `ngOnInit`.
Not the `spec` one.

We procede like this:

- Create a function `onSave` in the file.
- Get elements from the line
- call auction service to save the new auction
- retrieve informations
- add them to the auction array.
- reset input and select tags.

Adding auctions

Let's write the function onSave.

Here is what it should look like.

```
onSave($event) {  
  // Get elements from the line  
  ...  
  // Save elements and get result  
  this._auctions.saveNew(auctionTitle, auctionAuthor, auctionBody, auctionCategory).subscribe(...);  
}
```

This function is currently empty.

We will fill it step by step.

Adding auctions

How do we get informations?

You can get them from input elements with their ID.

```
var auctionTitle = ((document.getElementById("title") as HTMLInputElement).value);
```

Same kind of instruction for a select tag.

```
((document.getElementById("category") as HTMLSelectElement).value);
```

Get the 4 elements you need.

Adding auctions

Then the saveNew function.

Service is `_auction`

So we have to write a function saveNew in the services.

This function will send us back an Observable that we will have to process.

We will see this part when we will have save the item.

Adding auctions

Now, function saveNew.

It is in the auction service file, the ts one.

This function cannot have a call to the database.

But node server can. So we have to send a request to node server.

- We create a json object with the parameters
- we send a request to node server. A post request, anybody should not use this request
- send back result.

Adding auctions

Here is what the function may look like:

```
saveNew(auctionTitle: string, auctionAuthor: string, auctionBody: string, auctionCategory: string) {  
  var newObject = {"id": -1, "title": auctionTitle, "author": auctionAuthor, "body": auctionBody, "category": auctionCategory};  
  var result = this._httpClient.post(this._serverURL+"/addAuctions", newObject);  
  return result;  
}
```

Adding auctions

Now, we have to write the fonction in the node server script that saves the auction.

We have to:

- Get informations from the post request
- Send a request to the database to save the auction.
- Get the saved auction and send it back as a response.

```
app.post('/addAuctions', function(req, res) {
```

Adding auctions

How can we get informations from a request?

- For a REST call, we already know that a route looks like `/theRoute/:param1/:param2/...`
- If it is another call, you can get them from the `req` parameter of the function
 - for a get call, use **`req.params.theNameOfTheParameter`**
 - for a post call, use **`req.body.theNameOfTheParameter`**

```
var title = req.body.title;
```

Get the 4 informations: title, author, body and category

Adding auctions

Now, we have to save the informations.

- connect to database
- use a request to insert data. Category can be a negative value, so your request should depend on category's value.
- get ID for inserted value
- get auction with the inserted id. Return it.

Adding auctions

```
var client = new pg.Client(conString);
client.connect(function(err) {
  if (err) {...} else {
    // INSERT data and get id in return
    var theQuery;
    var values;
    if (category > 0) {
      theQuery = 'INSERT INTO item(title, author, body, category_id) VALUES ($1, $2, $3, $4) RETURNING id';
      values = [title, author, body, category];
    } else {
      theQuery = 'INSERT INTO item(title, author, body) VALUES ($1, $2, $3) RETURNING id';
      values = [title, author, body];
    }
    client.query(theQuery, values, function(err, result) {
      if (err) {...} else {
        var id = result.rows[0]['id'];
        var tableReq = 'SELECT item.*, name FROM item LEFT OUTER JOIN Category ON (item.category_id=category.id) WHERE';
        var values = [id];
        listAll(req, res, tableReq, values);
      }
    });
  }
});
```

Adding auctions

Ok. What have we currently done?

- HTML calls onSave when a click event occurs.
- onSave get informations, and calls saveNew in the auction services
- saveNew send a request to the node server with method post to save the auction
- node server get auction, save information and send auction back
- saveNew get auction and send it back.
- save... that is the missing part. It has to get the auction, add it to the list and reset fields.

Adding auctions

Node server returns a JSON object to saveNew.
Did you remember loadAll and the Observables?
node returns an Observable. saveNew too.

So we have to use subscribe to force evaluation.
Remember, it is an asynchronous process. So, if we call save new,
and then write instructions to process the result it will fail.
Informations processing have to be in the callback function in
subscribe.

Adding auctions

What do we have to do with subscribe?

- Get informations. We create an array, we get the first value.
- Add the value to the auctions array
- reset input fields.

Remember, everything have to be done in the subscribe call or it will fail due to asynchrone call.

Adding auctions

A bit of... data type manipulation.

Unfortunately auction is an Object. We define it as an object, not as an Array of ...

The result is that if we try to add a line, it fails. Compiler refuses because it is an Object, not an Array.

So we have to do a bit of manipulation to save data in a temp array and put it back in the Object.

Adding auctions

Reseting fields.

Remember the way you've got informations from the HTML file?

The way you get title, author, ...?

Instead of getting informations from the variable, you only have to put something inside. An empty string for example.

Adding auctions

So here is what you should have:

```
this._auctions.saveNew(auctionTitle, auctionAuthor, auctionBody, auctionCategory).subscribe((value) => {
  // Add result to the auction array
  var temp = new Array();
  Array.prototype.push.apply(temp, this.auctions);
  temp.push(value[0]);
  this.auctions = temp;
  // Reset fields
  ((document.getElementById("title") as HTMLInputElement).value) = "";
  ((document.getElementById("author") as HTMLInputElement).value) = "";
  ((document.getElementById("body") as HTMLInputElement).value) = "";
  ((document.getElementById("category") as HTMLInputElement).value) = "-1";
}, (error) => {
  console.log(error);
});
```

Adding auctions

And the result

8	Anime	JY Martin	Selling LODOSS war	electronics	
	Astro	JY Martin	Looking for a Goto for my telescope	electronics 	

List of items

Auction #	Auction Type	Seller	Description	Category	
2	Anime	JY Martin	Looking for the end of the Fairy Tail manga (> 126)	books	
3	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	books	
4	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	electronics	
5	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?	pets	
6	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	books	
7	Anime	JY Martin	Looking for full version of Daimachi		
8	Anime	JY Martin	Selling LODOSS war	electronics	
9	Astro	JY Martin	Looking for a Goto for my telescope	electronics	
				- 	

Removing auctions

Next button, removing auctions, the red trash on each line.

We have to:

- catch event when clicking on the button
- call a function with the auction ID. The function is in the component module
- call a service that will delete the auction
- call node server to delete auction in database

Removing auctions

First, node server.

Add a function in the node server script.

- it is called in post mode. We do not want it to be called in another way.
- route is `"/removeAuction"`
- it is not a REST call, so you will have to get auction id from the req parameter
- Have a look to `listAll` to implement a call to the database that removes an auction from database
- It returned a JSON object : `JSON.stringify({ok:1})`

implement this function.

Removing auctions

Next, the service one. Auction service.
Add a method:

- name is remove
- parameter is auctionId
- create a JSON object "{id:auctionId}"
- call node server in post mode, use route removeAuction and the JSON object as a parameter
- return result

Implement the method.

Removing auctions

Next, the component function.

- add a onDelete method with \$event and an auction as parameters. "auction" is the object we want to remove.
- call method remove from the auction service. Parameter is the "auction.id"
- subscribe will tell the auction is removed. So we only have to remove the line.
 - get index in the array
 - use method splice on Arrays to remove it.

```
// Remove auction.id from the auction array
var temp = new Array();
Array.prototype.push.apply(temp, this.auctions);
var index = temp.findIndex( element => element.id == auction.id);
if (index >=0) {
    temp.splice(index, 1);
}
this.auctions = temp;
```

Removing auctions

At last, the html call.

On each trash button, we call onDelete when a click event occurs. onDelete has 2 parameters, the event and the auction we want to delete.









```
><button (click)="onDelete($event, auction)"><
```

Removing auctions

Let's try.

Remove last one.

List of items

Auction #	Auction Type	Seller	Description	Category	
2	Anime	JY Martin	Looking for the end of the Fairy Tail manga (> 126)	books	
3	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	books	
4	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	electronics	
5	Kikou	M Servieres	My dog Kikou gave me plenty of little dogs, who wants one?	pets	
6	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	books	
7	Anime	JY Martin	Looking for a full version of Daimachi		
8	Anime	JY Martin	Selling LODOSS war	electronics	
	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	

Removing auctions

Or another one.

List of items

Auction #	Auction Type	Seller	Description	Category	
2	Anime	JY Martin	Looking for the end of the Fairy Tail manga (> 126)	books	
3	Elenium	JY Martin	I am looking for the french version of the The Elenium series By David Eddings	books	
4	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer	electronics	
6	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?	books	
7	Anime	JY Martin	Looking for a full version of Daimachi		
8	Anime	JY Martin	Selling LODOSS war	electronics	
				- 	

Updating auctions

What about updating an auction?

We need

- a button we can click on, that switches the line to input fields
- switch button to "save line" button
- when clicking on the "save line" button, save data to the database
- switch input tags to text
- switch button back to an "update" button.

Updating auctions

First, the easiest part, the node server that will save the update.
Add a route for updateAuctions.

Remember the way we added an auction?

Process the same way. The ID comes from the request. SQL request is UPDATE instead of INSERT.

Maybe it could be a good idea to have a look to the category. If it is negative, you should update category_id to NULL.

Implement method in the node server script.

Updating auctions

Next, the auction service. Add an update method. Parameters are the nearly the same as saveNew. Add auctionID as a parameter.

Script is the same as saveNew except when you set the ID for the object, and the route you call.

Implement method update in the auction service script.

Updating auctions

Next, the html file.

We need a button to edit elements, and another one to save.
Save button must not been displayed.

What else?

We should be able to switch between text and input / select elements.

Maybe we could set both and display only one.

Updating auctions

So, let's set the auction elements.

Here is what we could do for title. Do the same for author and body.

```
<td>
  <div id="title_{{ auction.id }}" style="display:inline">{{ auction.title }}</div>
  <div id="e_title_{{ auction.id }}" style="display:none">
    <input type="text" name="v_title_{{ auction.id }}" id="v_title_{{ auction.id }}" value="{{ auction.title }}" size="15" />
  </div>
</td>
```

Category is nearly the same, except it is a SELECT tag.

```
<td>
  <div id="cat_{{ auction.id }}" style="display:inline">{{ auction.name }}</div>
  <div id="e_cat_{{ auction.id }}" style="display:none">
    <select name="v_cat_{{ auction.id }}" id="v_cat_{{ auction.id }}">
      <option value="-1">--</option>
      <option *ngFor="let category of categories" value="{{ category.id }}">{{ category.name}}</option>
    </select>
  </div>
</td>
```

Updating auctions

And the buttons.

One button is displayed, the other one is hidden.

```
<td style="text-align:center">
  <button (click)="onDelete($event, auction)">
    
  </button>
  <button (click)="onEdit($event, auction)" style="display:inline" id="edit_{{ auction.id }}">
    
  </button>
  <button (click)="onUpdate($event, auction)" style="display:none" id="save_{{ auction.id }}">
    
  </button>
</td>
```

Updating auctions

Now, the more difficult part: the auction components scripts, onDelete and onUpdate.

First, let's write 3 utilities.

- First one, switchEditElement: it hides a text div and shows the input div. Parameters are the element name and the id value.

```
switchEditElement(name: string, id: string) {  
  var staticElement = document.getElementById(name+"_"+id);  
  var inputElement = document.getElementById("e_" + name+"_"+id);  
  staticElement.style.display = "none";  
  inputElement.style.display = "inline";  
}
```

- Second one, switchStaticElement: it hides an input div and shows the text div
- Third one, switchButton: switched a button display style from inline to none and none to inline

Updating auctions

the onEdit method switches static text to editable text.
It should look like this:

```
onEdit($event, auction) {  
  // Get ID  
  var auctionId = auction.id;  
  
  // switched line elements  
  this.switchEditElement("title", auctionId);  
  this.switchEditElement("author", auctionId);  
  this.switchEditElement("body", auctionId);  
  this.switchEditElement("cat", auctionId);  
  
  // switches buttons  
  this.switchButton("edit", auctionId);  
  this.switchButton("save", auctionId);  
}
```

Updating auctions

Now the onUpdate method:

```
onUpdate($event, auction)
```

- gets informations from input elements, like we did to save a new auction,
- sends them to the service,
- Get information telling it is done
- Set text elements to the input elements
- Switches line to static text
- Switches buttons display back to their initial value

Updating auctions

- Adding information to the array value:

```
this._auctions.update(auctionId, auctionTitle, auctionAuthor, auctionBody, auctionCategory).subscribe((value) => {  
  // Add result to the auction array  
  var temp = new Array();  
  Array.prototype.push.apply(temp, this.auctions);  
  var index = temp.findIndex( element => element.id == auction.id);  
  if (index >=0) {  
    temp.splice(index, 1, value[0]);  
  }  
  this.auctions = temp;  
});
```

Updating auctions

- Reseting fields

Maybe you could write a function that sets a static text in the div element

```
setDivText(name: string, id: string, value: string) {  
  var staticElement = document.getElementById(name+"_"+id);  
  var textDiv = document.createTextNode(value);  
  while (staticElement.firstChild !== null) {  
    staticElement.removeChild(staticElement.firstChild);  
  }  
  staticElement.appendChild(textDiv);  
}
```

And reset elements in the line.

Updating auctions

Ok, let's try it.

8	Anime	JY Martin	Selling LODOSS war
---	-------	-----------	--------------------

Click on the button.

8	<input type="text" value="Anime"/>	<input type="text" value="JY Martin"/>	<input type="text" value="Selling LODOSS war"/>
---	------------------------------------	--	---

Updating auctions

Change value.

8	Anime	JY Martin	Selling LODOSS war DVD
---	-------	-----------	------------------------

Save it.

8	Anime	JY Martin	Selling LODOSS war DVD
---	-------	-----------	------------------------

Updating auctions

Want to be sure?
Refresh screen.

List of items

Auction #	Auction Type	Seller	Description
2	Anime	JY Martin	Looking for the end of the Fairy Tail manga (> 126)
3	Elenium	JY Martin	I am looking for the french version of the The Elenium series By Dav
4	Kinect	JM Normand	I sell my new kinect that I can't connect to my computer
6	Mangas	M Magnin	I am looking for the first Alabator Mangas. Anyone get it?
7	Anime	JY Martin	Looking for a full version of Daimachi
8	Anime	JY Martin	Selling LODOSS war DVD

Connection screen.

Ok, we can add, update and remove an auction.
What next?

The connection screen.

Connection screen.

We will use another way of managing data: object classes.

- Create a new component: authenticate.
- Use materials to add html and css elements
- Change default route to authenticate
- Create service authenticate
 - add method checkAuthenticate with 2 parameters, login and password. check for admin / admin
 - reply with a JSON object. 1 for true, 0 for false.

Connection screen.

We have to change a little bit the html file to ask angular to manage elements.

- For each bloc entry element, we define a **FormGroup** angular element to manage it.
- we add a control attribute in each entry tag: `formControlName`
- when submitting, we ask angular to call our component.

```
<form (ngSubmit)="authenticateUser()">
  <h1>Auctions Login</h1>
  <p [formGroup]="loginData"><input type="login" name="user" placeholder="Login"
    formControlName="login" />
  </p>
  <p [formGroup]="passwdData"><input type="password" name="passwd" placeholder="Password"
    formControlName="passwd" /></p>
  <p><button type="submit">Login</button></p>
</form>
```

Connection screen.

To manage forms we use 2 modules:

- ReactiveFormsModule
- FormsModule

So, in the app.module.ts file, add the import:

```
import { ReactiveFormsModule, FormsModule } from '@angular/forms';
```

And add them to the imports array.

Connection screen.

Next, the ts element.

- we add attributes to manage data. We will use `ngOnInit` for that.
 - `FormGroups` to manage the elements
 - `FormControls` to define control functions
- our `authenticateUser` method.

Connection screen.

Here is what it looks like.

```
export class AuthenticateComponent implements OnInit {  
  loginData: FormGroup;  
  passwdData: FormGroup;  
  login: FormControl;  
  passwd: FormControl;  
}
```

You will need to import elements from angular forms.

```
import { FormGroup, FormControl } from '@angular/forms';
```

Connection screen.

We create FormGroup and FormControl in ngOnInit.

```
ngOnInit() {  
  this.loginData = new FormGroup({  
    login : new FormControl()  
  });  
  this.passwdData = new FormGroup({  
    passwd : new FormControl()  
  });  
}
```

FormGroup names have to be the same as the ones in the html file.

Connection screen.

Now, the authenticateUser method.

To understand what happens, we start with the console.

```
authenticateUser() {  
  console.log(this.loginData);  
  console.log(this.passwdData);  
}
```

Check the angular server is launched and connect to
<http://127.0.0.1:4200>

Use dev tools to check console. You should have the 2 attributes
with values.

Connection screen.

Auctions Login

Login

Password

Login

```
▼ FormGroup
  ▶ f _onCollectionChange: function()
  ▶ O _onDisabledChange: [] (0)
  ▶ N asyncValidator: null
  ▶ O controls: {login: FormControl}
  ▶ N errors: null
  ▶ B pristine: false
  ▶ S status: "VALID"
  ▶ O statusChanges: EventEmitter {_isScalar: false,
  ▶ B touched: true
  ▶ N validator: null
  ▶ O value: {login: "test"}
  ▶ O valueChanges: EventEmitter {_isScalar: false,
```

Connection screen.

Ok, now let's really implement it.

- first import authenticate service
- next, in the constructor, create an attribute for AuthenticateService. Have a look to AuctionComponent if you have a problem.
- you get login string value with:
`var loginValue = this.loginData.value.login;`
- Same for passwd
- get checkAuthenticate result from your AuthenticateService attribute with the login and password.
- use `JSON.parse` on the result to get an object with the result.
- switch to auction page if it is ok.

Connection screen.

How do i switch to another page?

1 - import Router module

```
import { Router } from '@angular/router';
```

2 - create a Router attribute in the constructor. Let's call it router.

3 - ask the router to go to the "auctions" route

```
this.router.navigate(["auctions"]);
```

Connection screen.

Auctions Login

Auctions Login

Auctions Login

List of items

Auction #	Auction Type	Seller
2	Anime	JY Martin
3	Flenium	JY Martin

