TD5 : Http Communication between Angular (Frontend) and NodeJs (Backend)

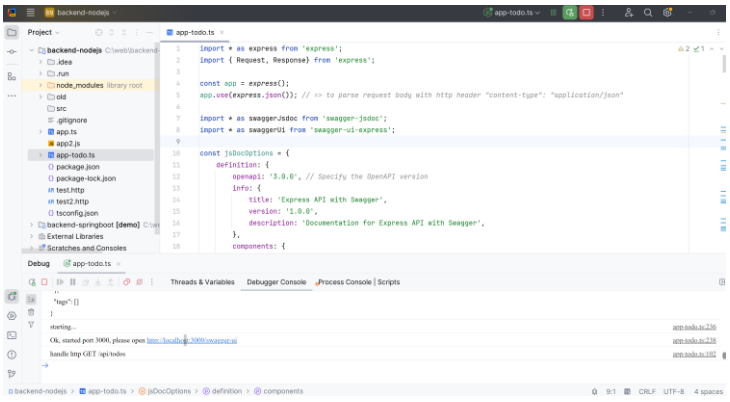
Pre-requisites: your previous TDs

- a working Angular (Frontend) project
- AND a working NodeJs-Express (Backend) project, exposing a Rest API for a CRUD object (either the canonical “Todo” app or “LearningPackage” for anki-like app)

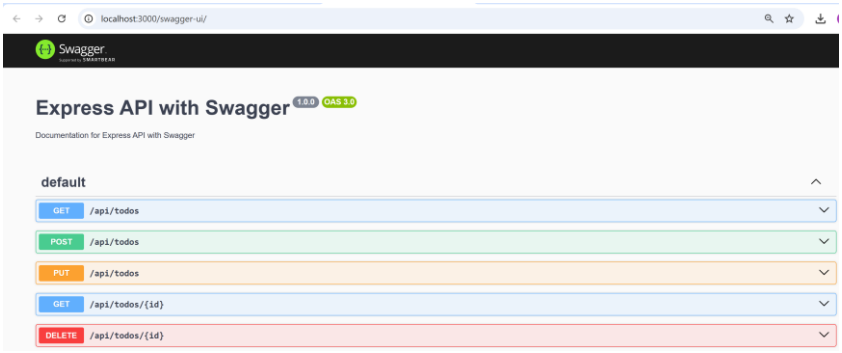
The goal of this TD is to make them communicate: the Angular client will call http POST,PUT,GET,DELETE to the backend to get all its data, and perform update actions on them.

It is preferrable that the server really save data in a postgresql database, but the TD can still be done using in-memory objects.

Step 1: Reopen your backend project in your IDE, and launch it.

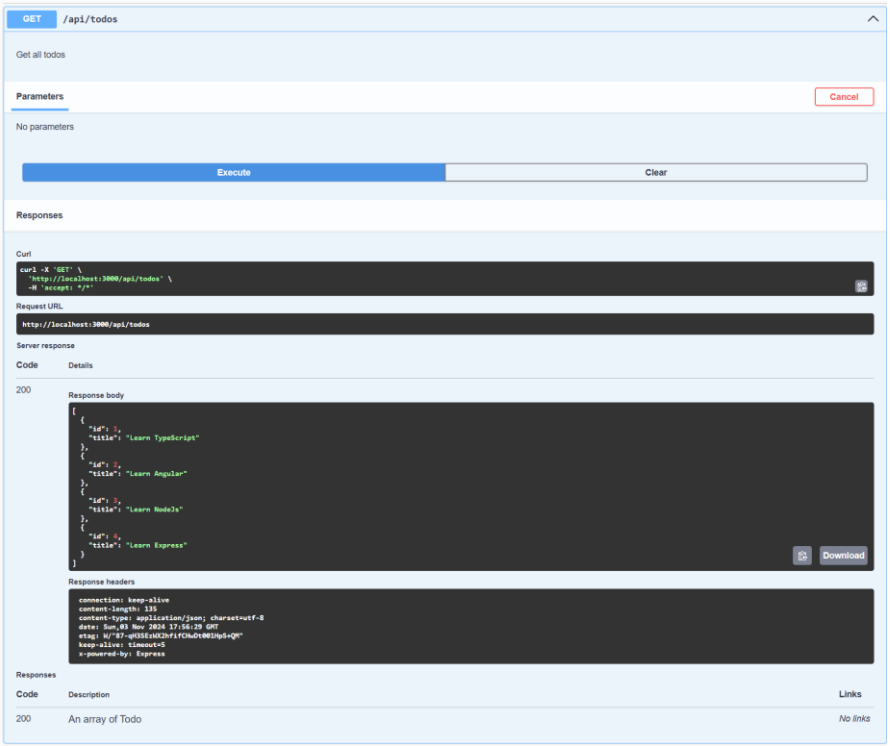


Your backend should expose all CRUD methods: POST to Create, GET to Read, PUT to Update, DELETE to Delete items.

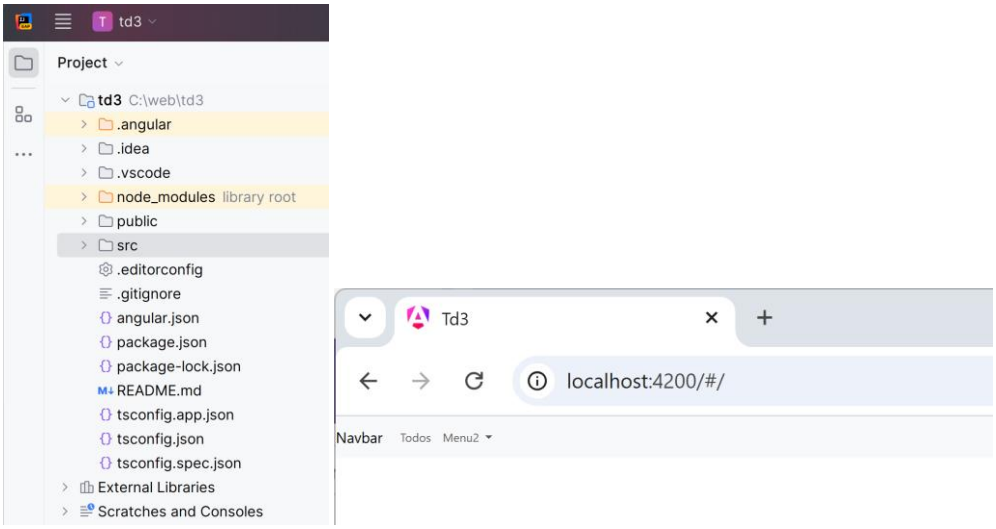


You should be able to test all methods are working correctly, either using Swagger, Curl, Postman, Intellij ".http" files, or any other tool.

Example testing GET "/api/todos", using Swagger:



Step 2: Re-open your Angular project in your IDE and launch it



Step 3: setup classes

Create an Angular page component, to view the list of items

ng g c todo-list-page

(or ng g c learningpackage-list-page)

Bind this page to a router-outlet URL, for example "/todos",

```
export const routes: Routes = [
    {path:'todos', component: TodoListPageComponent},
```

and create a menu item in your application menu.

```
<li class="nav-item">
  <a class="nav-link" routerLink="/todos">Todos</a>
</li>
```

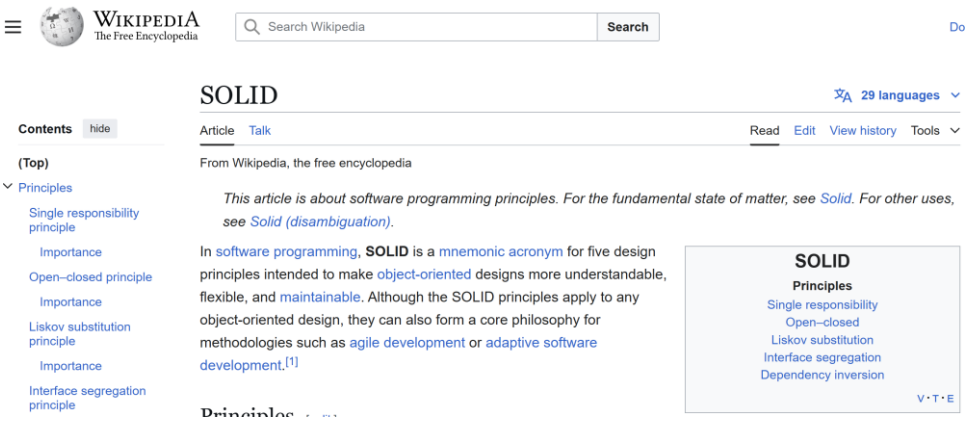
Create a corresponding service,

ng g s todo

Inject the service in the component TodoListPageComponent (either using constructor injection, or angular 17 "inject()" method to initialize a private readonly field)

```
import {Component, inject} from '@angular/core';
import {TodoService} from '../todo.service';

..

export class ListPageComponent {
  private readonly todoService = inject(TodoService);
```

The service is to put all the Http code to call the service. The view components should NOT contain code related to Http code (nor any business logic code), only "view" code, following the **Model-View-Controller** design pattern, and the **SOLID** (Single Responsibility-OpenClose-Liskov-Interface-Dependency Injection principles).. see https://en.wikipedia.org/wiki/SOLID



The service should expose method to get list of Todos, update Todo, etc... as does the CRUD Rest API of the backend, but all return types will be wrapped with RxJs “Observable<>”

Example to get a list of todos:

```
import {Observable, Subject} from 'rxjs';


getTodos(): Observable<TodoModel[]> {  // using RxJs Observable
    ...
```

instead of

```
getTodos():TodoModel[] {    // wrong
```

Concerning the “Model” (M of the MVC), it should be a class rather than a Typescript “interface”, and preferably immutable (it is not always practical, but when it is, the code is safer and cleaner).

From your nodeJs project, copy the interface definition corresponding to the “DTO” (Data Transfer Object) exchange JSON format:

```
export // temporary for test only
interface TodoDTO {
  id: number;
  title: string;
  description?: string;
```

```
  priority?: number;
}
```

This interface is not supposed to be used except in the service, while communicating with the Rest API.

Create a corresponding model class, that is exported:

```
export class TodoModel {
 id: number;
 title: string;
 description: string;
 priority: number;

 constructor(src: TodoDTO) {
  this.id = src.id || -1;
  this.title = src.title || '';
  this.description = src.description || '';
  this.priority = src.priority || 1;
 }
}
```

Step 4: Setup HTTP client Provider in app.config.ts

```
import {provideHttpClient} from '@angular/common/http';

export const appConfig: ApplicationConfig = {
 providers: [
  .... <truncated>
  provideHttpClient()   // ← line to add
 ]
};
```

Step 5: Write the Angular httpClient .get / .put /.post methods (first using DTO interface… refactored later to use Model classes)

```
import {HttpClient} from '@angular/common/http';
```

```
...
private readonly httpClient = inject(HttpClient);
...
getTodoDTOs(): Observable<TodoDTO[]> {
   return this.httpClient.get<TodoDTO[]>('/api/todos');
}
```

Step 6: call this code from the component, subscribe to the asynchronous result, and save it to a field

```
<button (click)="onClickReload()" class="btn btn-small btn-outline-info">Load
Todos</button>

<ul>
 <li *ngFor="let todo of todos">
  {{todo.id}}
 </li>
</ul>
```

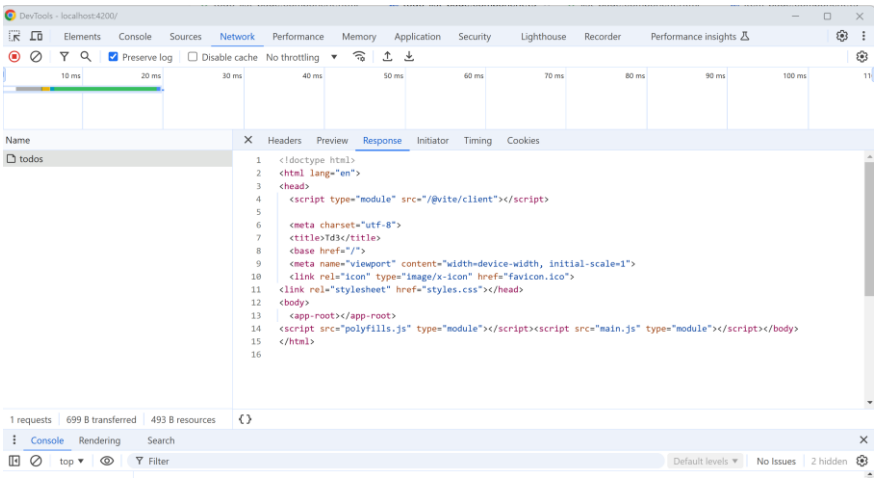
In todo-list-page.component.ts :

```
@Component({
 selector: 'app-todo-list-page',
 standalone: true,
 imports: [
  NgForOf
 ],
 templateUrl: './todo-list-page.component.html'
})
export class TodoListPageComponent {
        readonly todoService = inject(TodoService);

        todos: TodoDTO[] = [];

        onClickReload() {
```

```
    this.todoService.getTodoDTOs().subscribe({
      next: data => {
        console.log('finished loaded Todos, saving to component field');
        this.todos = data;
      }, error: err => {
        console.log('Failed to load Todos from Http server', err);
      }
    })
  }
```

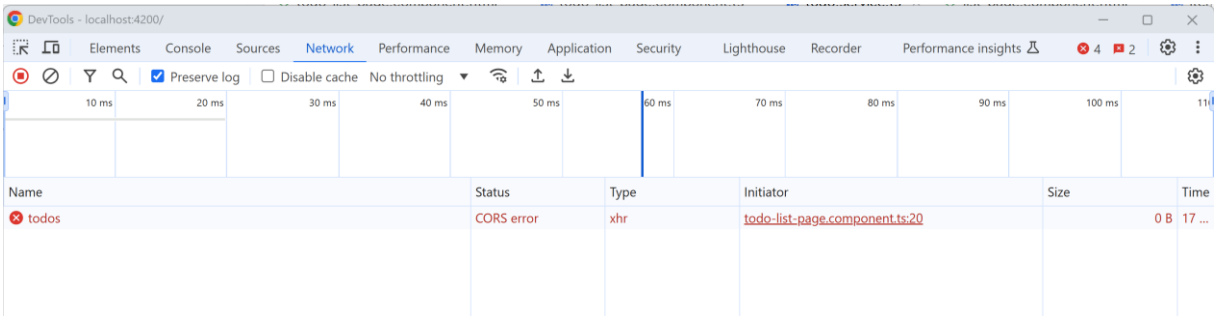
Step 7: test it … it will fail (the backed is NOT responding to the port http://localhost:4200 yet, but only to http://localhost:3000)

check the error in Chrome DevTools



Step 8: FALSE good idea … change the called URL to be http:/localhost:3000 … check that it fail with CORS Origin problem

```
  getTodos(): Observable<TodoDTO[]> {
    return this.httpClient.get<TodoDTO[]>('http://localhost:3000/api/todos');
                                          // ←== BAD, and causing Cors ERROR …
  }
```

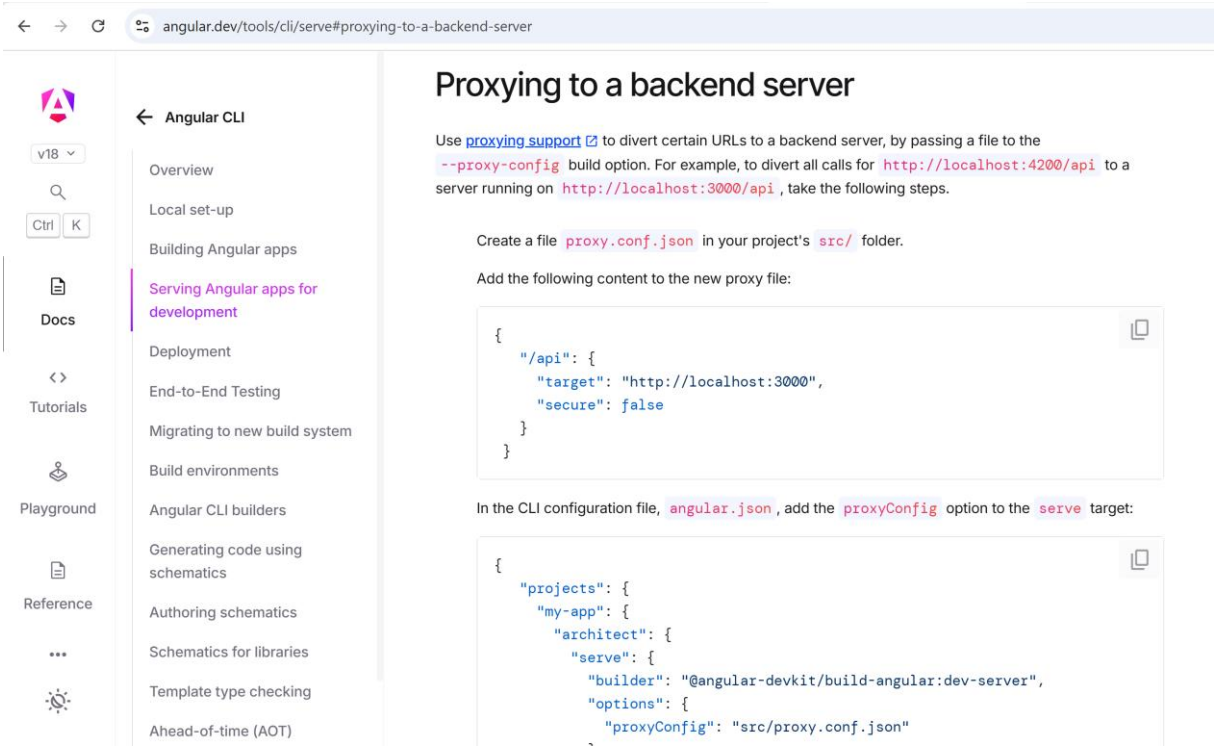


Step 8: Solve the CORS origin problem by adding an angular proxy from http://localhost:4200/api/* to http://localhost:3000/api

First revert your temporary trial code from ste8, to be relative URL again (relative to http://localhost:4200)

```
getTodos(): Observable<TodoDTO[]> {
 return this.httpClient.get<TodoDTO[]>('/api/todos');
}
```

See official Angular doc: https://angular.dev/tools/cli/serve#proxying-to-a-backend-server



Create a file “src/proxy.conf.json”

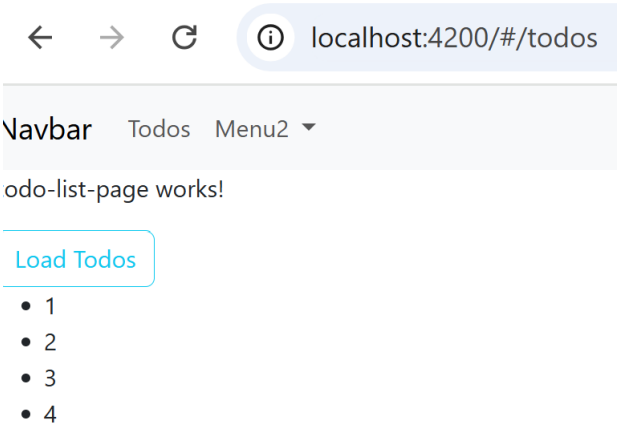
```
{
  "/api": {
    "target": "http://localhost:3000",
    "secure": false
  }
}
```

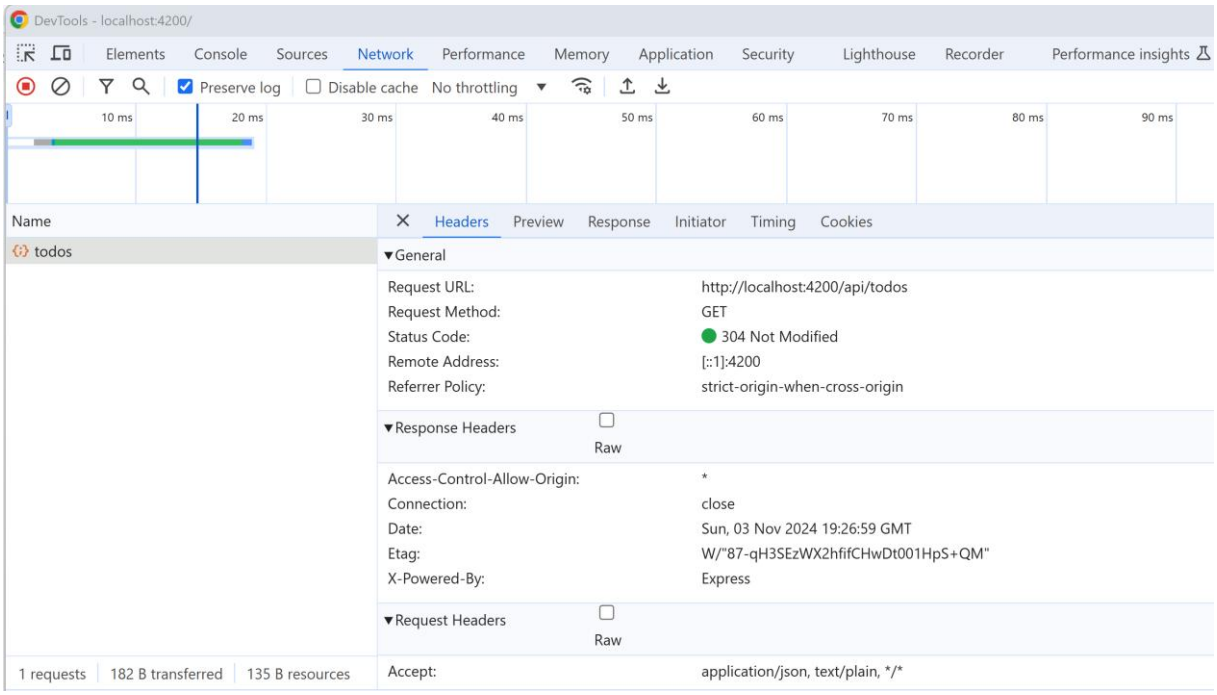
Then edit file “angular.json” to add this line in the “architect > serve > options” (or “architect > serve > development > options”)

```
"proxyConfig": "src/proxy.conf.json"
```

Stop and Relaunch “ng serve”

Check that it works





Step 9: Refactor the code to return Model from the Service, using ".pipe(map(...))"

As said previously, the service should not return "TodoDTO" but "TodoModel" instead.

You can call the "map(function)" operator, and passing it a function that take "TodoDTO[]" and return a "TodoModel[]"

For example:

```
toDTOs(todoArray: TodoDTO[]): TodoModel[] {
    return todoArray .map(x => new TodoModel(x));
}
```

The map Operator must be put in a ".pipe(...)"

This gives:

```
getTodoDTOs(): Observable<TodoModel[]> {
  return this.httpClient.get<TodoDTO[]>('/api/todos')
    .pipe(map(todoArray => this.toDTOs(todoArray)))
}
```

Equivalently, in 1 line...

```
getTodoDTOs(): Observable<TodoModel[]> {
  return this.httpClient.get<TodoDTO[]>('/api/todos')
```

```
    .pipe(map(src => src.map(x => new TodoModel(x))))
}
```

Step 10: use swagger OpenAPI to generate all the Json "DTO" interfaces, and the Http clients method endpoints

First, modify your backend NodeJs if it is not yet exposing the "swagger.json" document in http GET

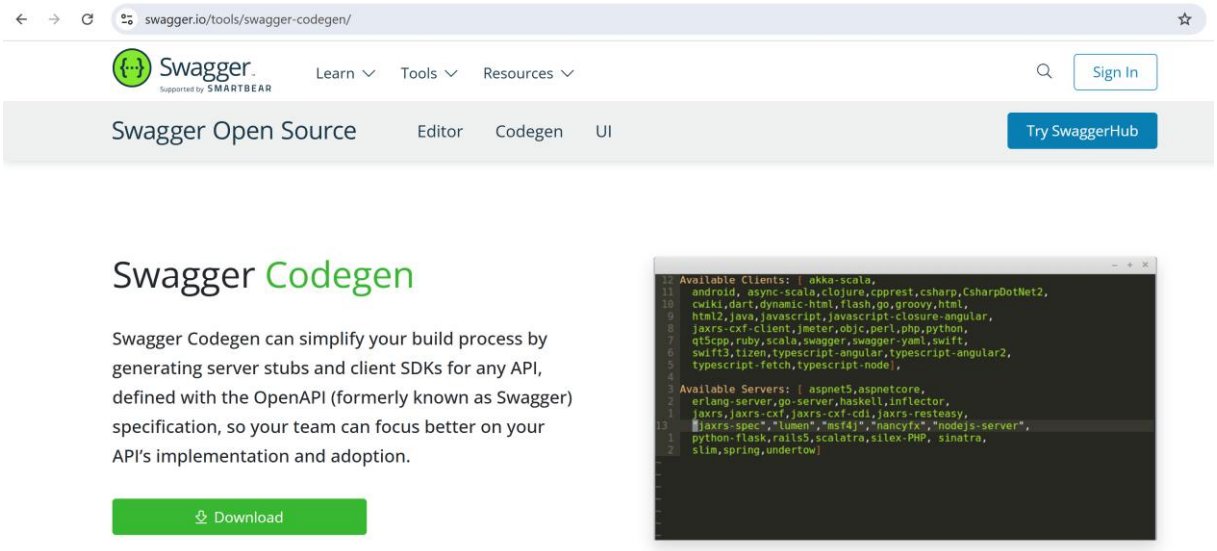
```
app.use('/swagger-ui', swaggerUi.serve, swaggerUi.setup(apiDoc));

app.get('/swagger.json', (req, res) => {
  res.setHeader('Content-Type', 'application/json');
  res.send(apiDoc);
});
```

Relaunch you NodeJS backend, and check you can get the "http://localhost:3000/swagger.json"

Save it to a local file "swagger.json"

Then open the Swager CodeGen (Code Generator):

https://swagger.io/tools/swagger-codegen/



The download button is broken ..o the source code repo !!

ChatGPT can help you

show me the java command line to generate typescript-angular code using java swager-codegen-cli and swagger.json spec

To generate a TypeScript-Angular client using the `swagger-codegen-cli` with a `swagger.json` specification file, you can use the following command:

bash Copy code

```
java -jar swagger-codegen-cli.jar generate \
   -i swagger.json \
   -l typescript-angular \
   -o ./generated/angular-client \
   --additional-properties ngVersion=12
```

Dowload the jar found in maven central repo: (use https://search.maven.org)

https://repo1.maven.org/maven2/io/swagger/codegen/v3/swagger-codegen-cli/3.0.63/

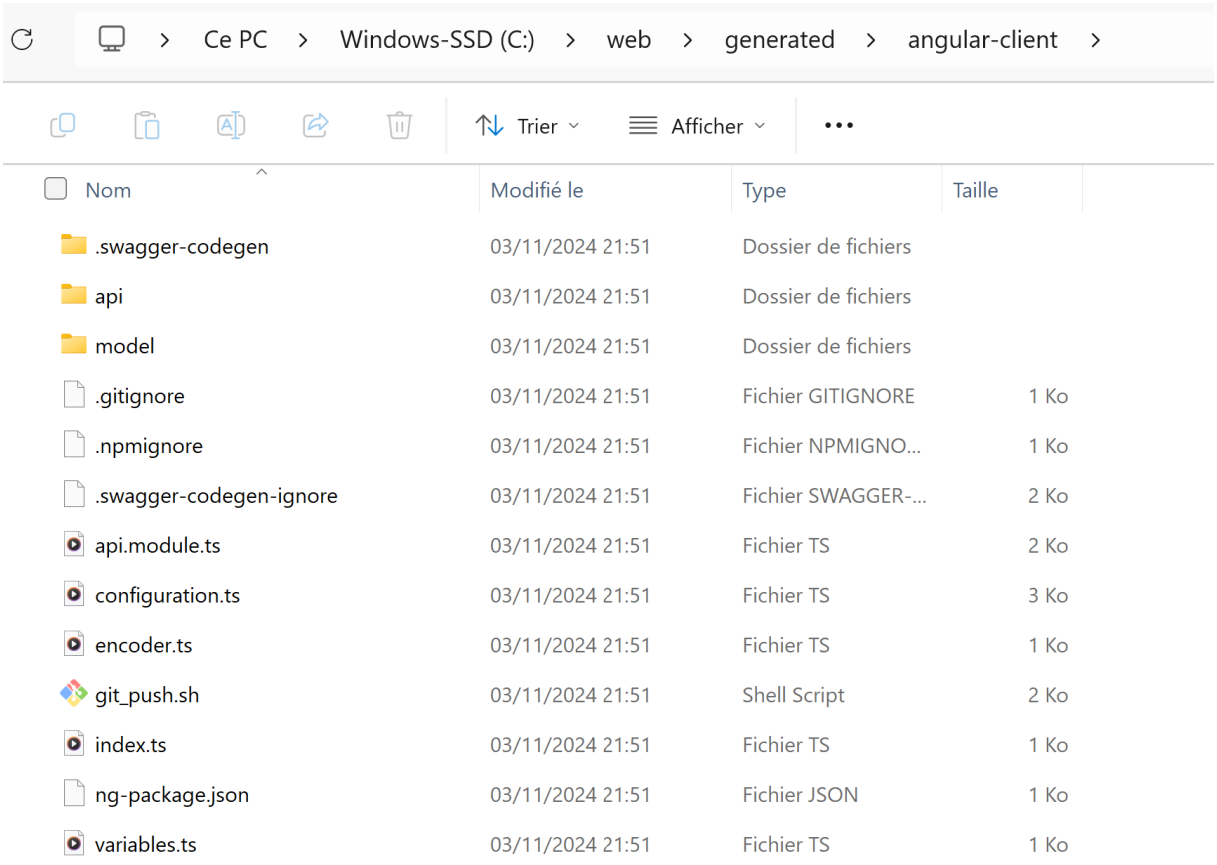
download the jar file, then launch java command to see parameters

c:> java -jar swagger-codegen-cli-3.0.63.jar generate -h

```
$ java -jar swagger-codegen-cli-3.0.63.jar generate -h
usage: swagger-codegen generate [-h] [-v []] [-l] [-o] [-i] [-t]
                       [--template-version] [--template-engine] [-a] [-c]
                       [-D [ [ ...]]] [-s] [--api-package]
                       [--model-package] [--model-name-prefix]
                       [--model-name-suffix]
                       [--instantiation-types [ [ ...]]]
                       [--type-mappings [ [ ...]]]
                       [--additional-properties [ [ ...]]]
                       [--import-mappings [ [ ...]]]
                       [--ignore-import-mapping []] [--invoker-package]
                       [--group-id] [--artifact-id] [--artifact-version]
                       [--library] [--git-user-id] [--git-repo-id]
                       [--release-note] [--http-user-agent]
                       [--reserved-words-mappings [ [ ...]]]
                       [--ignore-file-override]
```

java -jar swagger-codegen-cli.jar generate -i swagger.json -l typescript-angular -o ./generated/angu lar-client --additional-properties ngVersion=17

```
$ java -jar swagger-codegen-cli.jar generate -i swagger.json -l typescript-angular -o ./gener
ated/angular-client --additional-properties ngVersion=17
21:51:10.434 [main] INFO  i.s.c.v.g.t.AbstractTypeScriptClientCodegen - Template folder: null
21:51:10.436 [main] INFO  i.s.c.v.g.t.AbstractTypeScriptClientCodegen - Template engine: io.sw
agger.codegen.v3.templates.HandlebarTemplateEngine@12dae582
21:51:10.529 [Thread-0] INFO  i.s.c.v.g.t.AbstractTypeScriptClientCodegen - Template folder: n
ull
21:51:10.530 [Thread-0] INFO  i.s.c.v.g.t.AbstractTypeScriptClientCodegen - Template engine: i
o.swagger.codegen.v3.templates.HandlebarTemplateEngine@40f5816e
21:51:10.613 [Thread-0] WARN  i.s.c.v.i.CodegenIgnoreProcessor - Output directory does not exi
st, or is inaccessible. No file (.swagger-codegen-ignore) will be evaluated.
21:51:10.927 [Thread-0] INFO  i.s.codegen.v3.AbstractGenerator - writing file C:\web\.\generat
ed\angular-client\model\todo.ts
21:51:10.967 [Thread-0] INFO  i.s.codegen.v3.AbstractGenerator - writing file C:\web\.\generat
```

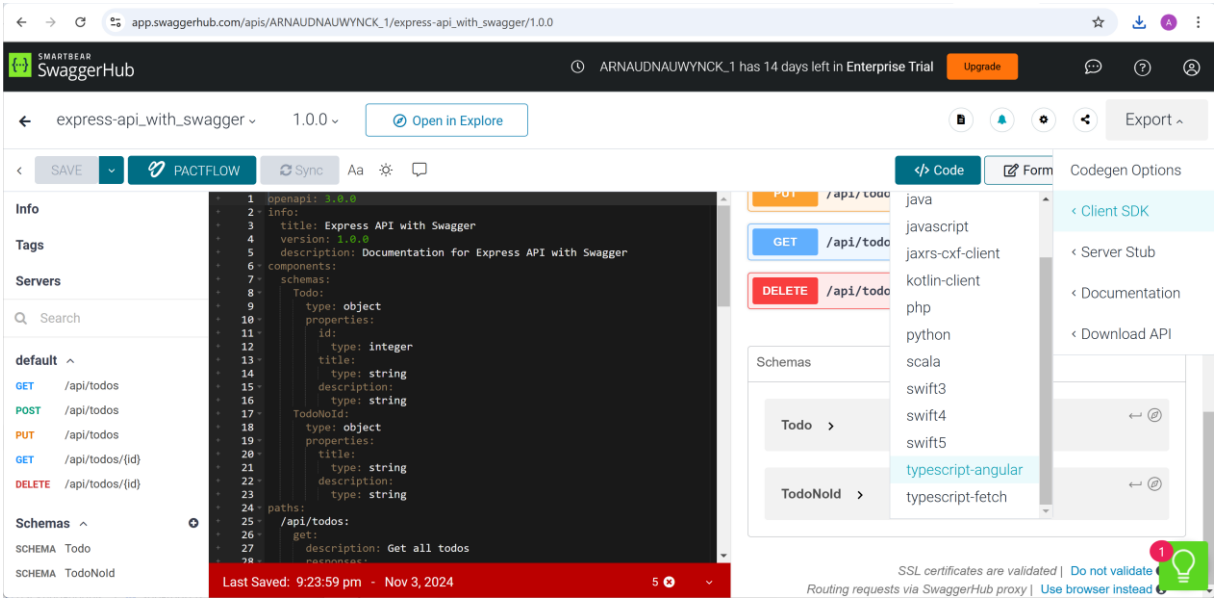


Alternative Step 10 b/: Regenerate your Angular client code, using swagger maven plugin

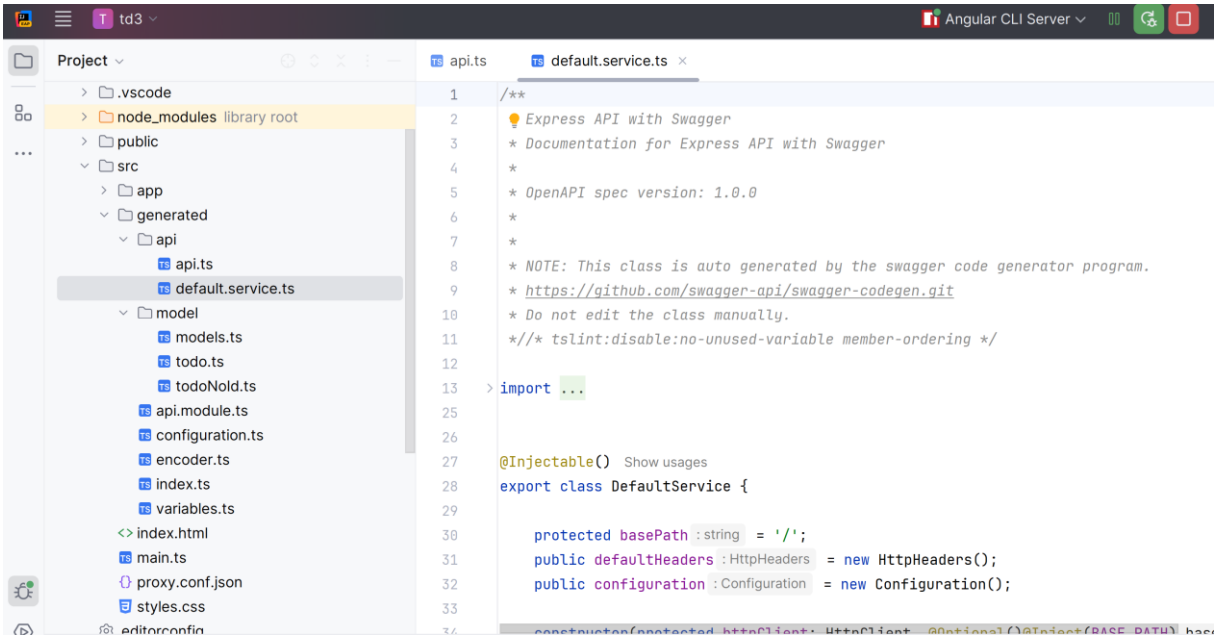
Cf course ...

Alternative Step 10 c/: generate using “Try SwaggerHub” (Online version)

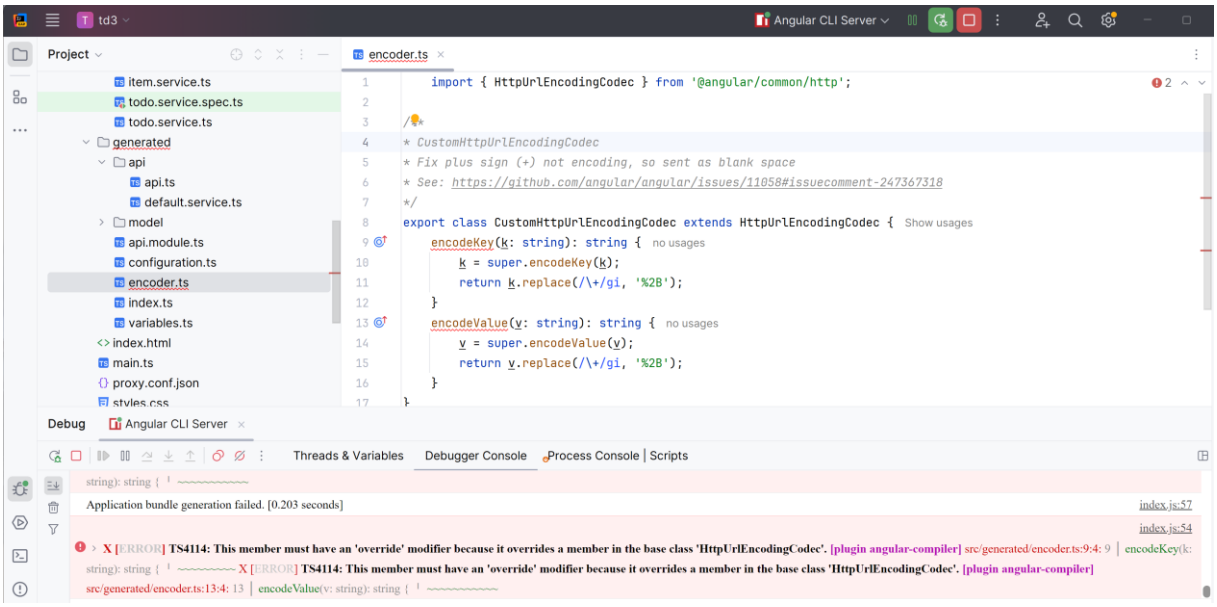
First login, then import you swagger.json file, and export > Client SDK > typescript-angular



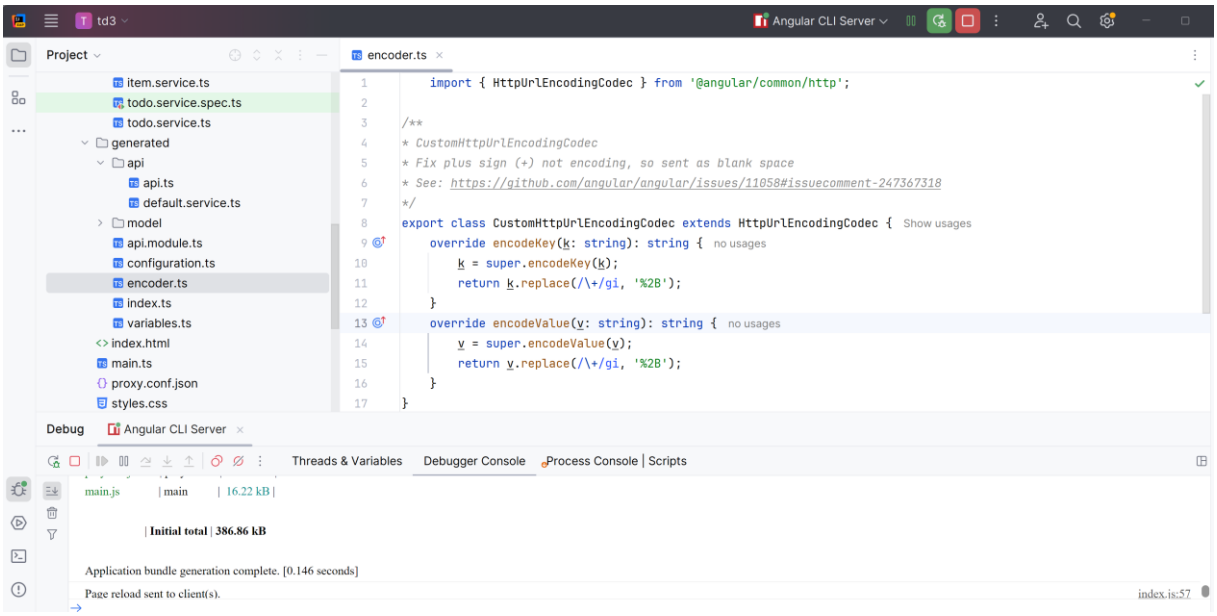
Step 11 *copy the code generated from generated/angular-client to your Angular src/*



Fix the error in the generated code: missing override



Add missing “override” on the 2 methods of file “encoder.ts” as below



Step 12: configure your angular project to use legacy module (as in angular version < 17)

In file app.config.ts, add the part in bold :

*import {ApplicationConfig, **importProvidersFrom**,*
provideZoneChangeDetection} from '@angular/core';
import {ApiModule, Configuration, ConfigurationParameters} from
'../generated';

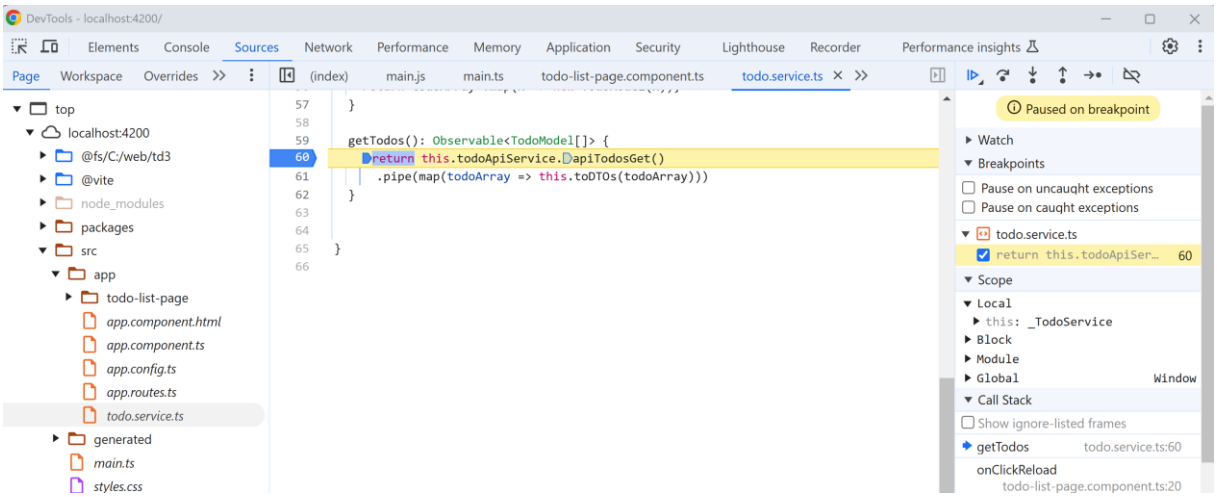
```
const apiConfParams : ConfigurationParameters = {
 basePath: 'http://localhost:4200/' // override generated code
}

export const appConfig: ApplicationConfig = {
 providers: [
  ...  // truncated code
   importProvidersFrom(
       ApiModule.forRoot(() => new Configuration(apiConfParams))
   )
 ]
};
```

Step 14: replace your httpClient code by your generated API servive (DefaultService)

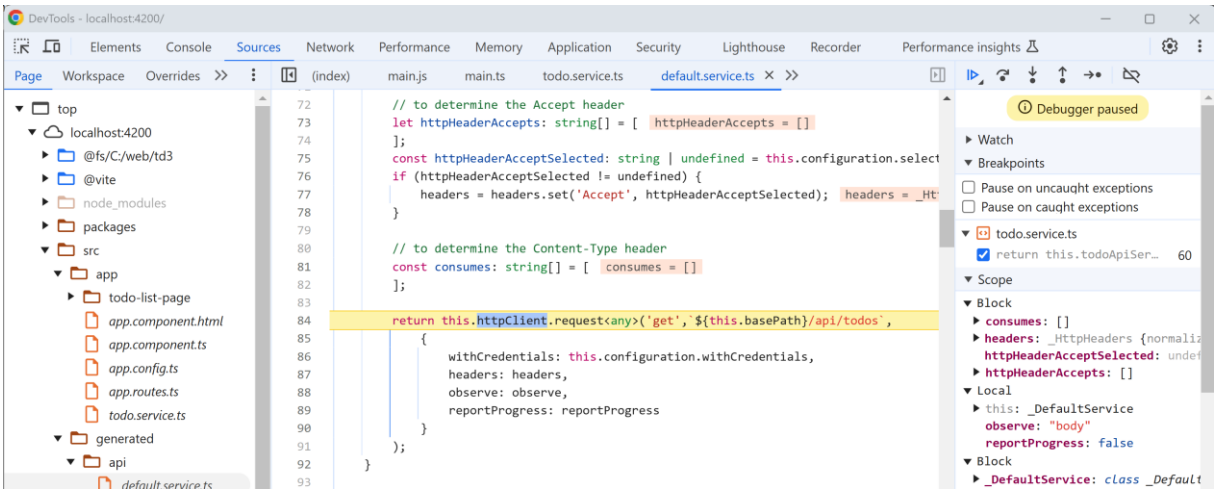
In file

```
import {DefaultService} from '../generated';

...

private readonly httpClient = inject(HttpClient);  // now unused, cf below

private readonly todoApiService = inject(DefaultService);


getTodos(): Observable<TodoModel[]> {
 return this.todoApiService.apiTodosGet()
   .pipe(map(todoArray => this.toDTOs(todoArray)))
}
```

Step 15: check that it still works ... debug in Chrome Dev Tools



As you can see by debugging “step into”, the generated code is just doing plain old “httpClient.request(‘get’, …)”



Now you can refactor safely code in backend, and regenerate frontend code… Your app will be compiled ahead of time to detect URL and JSON changed errors! This really make a huge difference for maintaining apps source code efficiently.