



Laboratorio 6

Construcción de un Cliente ‘grueso’ con un API REST, HTML5, JavaScript y CSS3. Parte I

url repositorio:

https://github.com/ARSW2023-2/ARSW_Lab06.git

Integrantes:

Luisa Fernanda Bermúdez Girón

Karol Daniela Ladino Ladino

Squad:

Inside Out

Profesor:

Javier Iván Toquica Barrera

Curso:

ARSW – 1

Fecha De Entrega:

06-10-2023

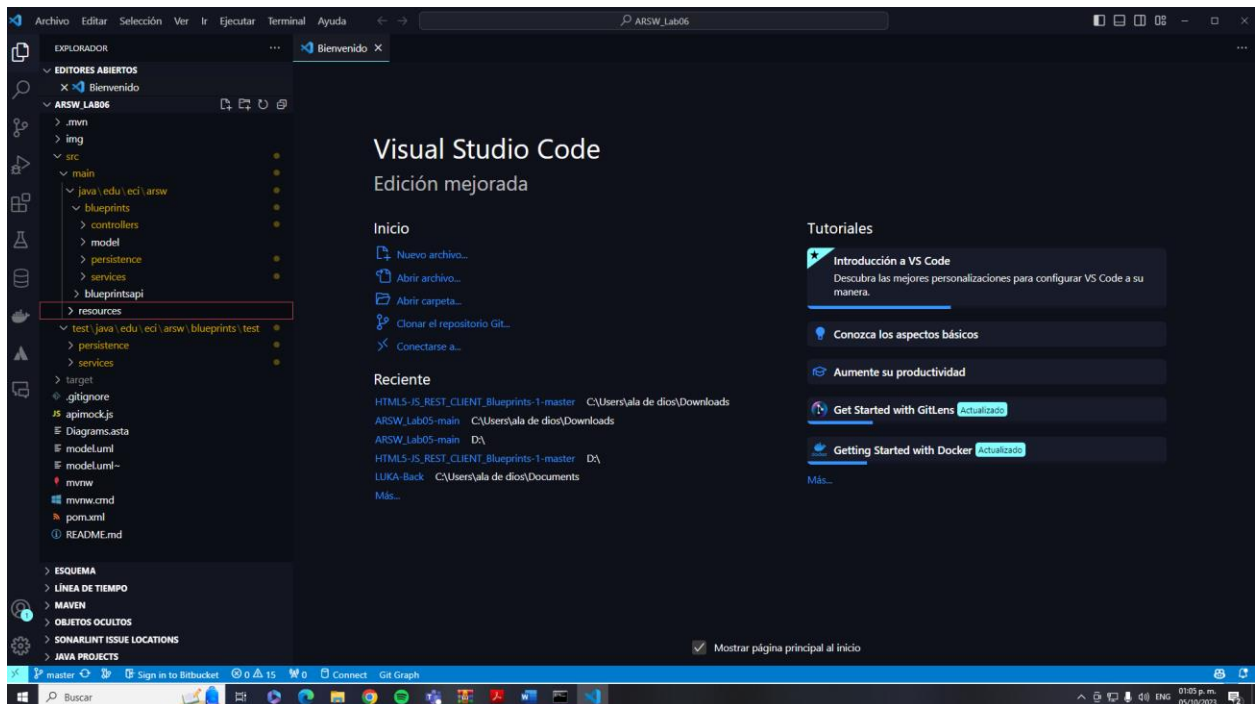
I. Introducción

En el campo de las Arquitecturas de Software, la construcción de aplicaciones web que interactúan con APIs REST se ha vuelto fundamental para proporcionar a los usuarios una experiencia eficiente y atractiva. Este laboratorio se centra en el desarrollo de un cliente "grueso" que utiliza tecnologías web modernas como HTML5, JavaScript y CSS3 para interactuar con un servicio de consulta de planos de autor. La aplicación permitirá a los usuarios buscar y visualizar planos, ofreciendo información detallada sobre los mismos.

II. Desarrollo del laboratorio

Ajustes Backend

1. Trabaje sobre la base del proyecto anterior (en el que se hizo el API REST).



- Incluya dentro de las dependencias de Maven los 'webjars' de jQuery y Bootstrap (esto permite tener localmente dichas librerías de JavaScript al momento de construir el proyecto):

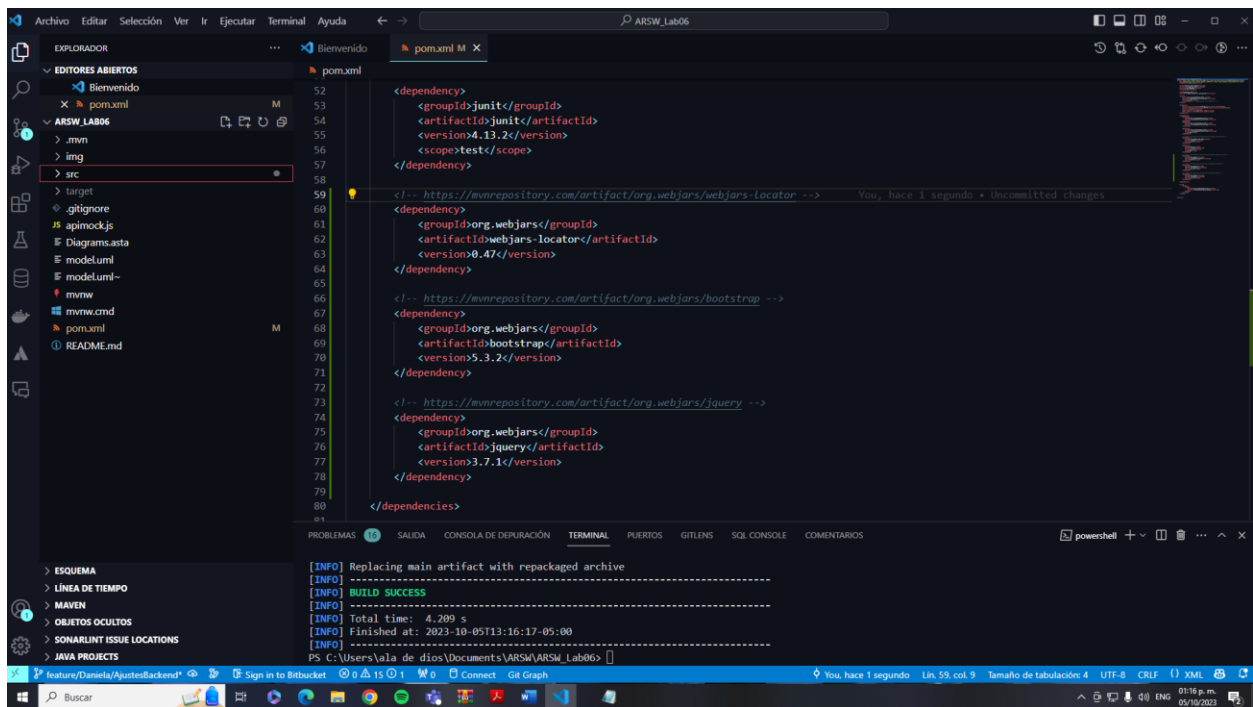
```
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>webjars-locator</artifactId>
</dependency>

<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>bootstrap</artifactId>
  <version>3.3.7</version>
</dependency>

<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>jquery</artifactId>
  <version>3.1.0</version>
</dependency>
```

➤ Modificación pom

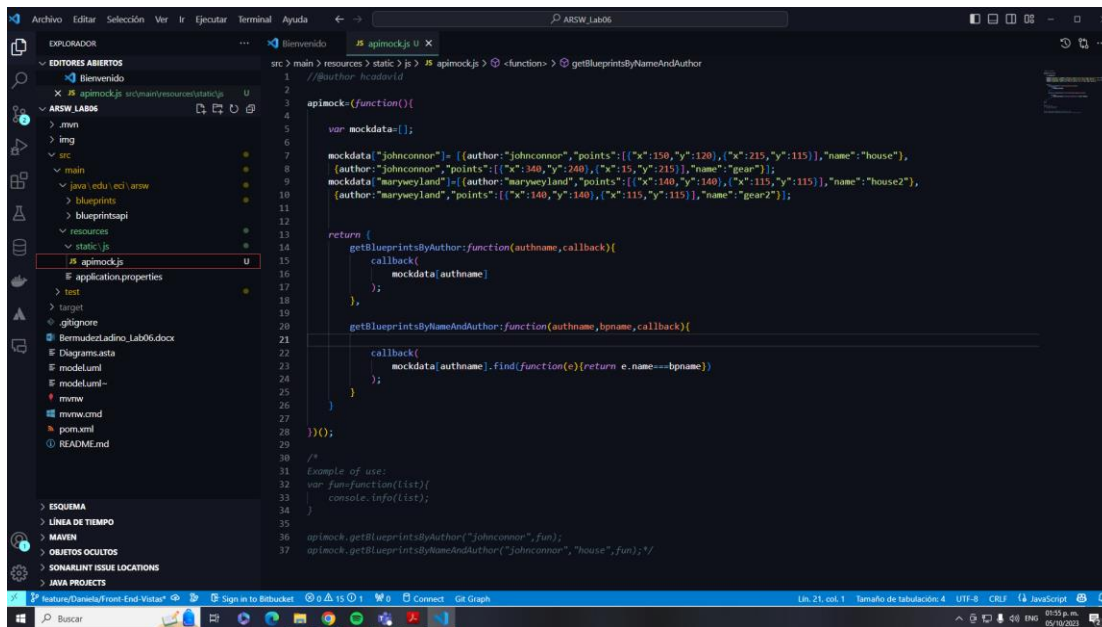
Incluimos las dependencias mencionadas con las ultimas versiones de cada una de estas



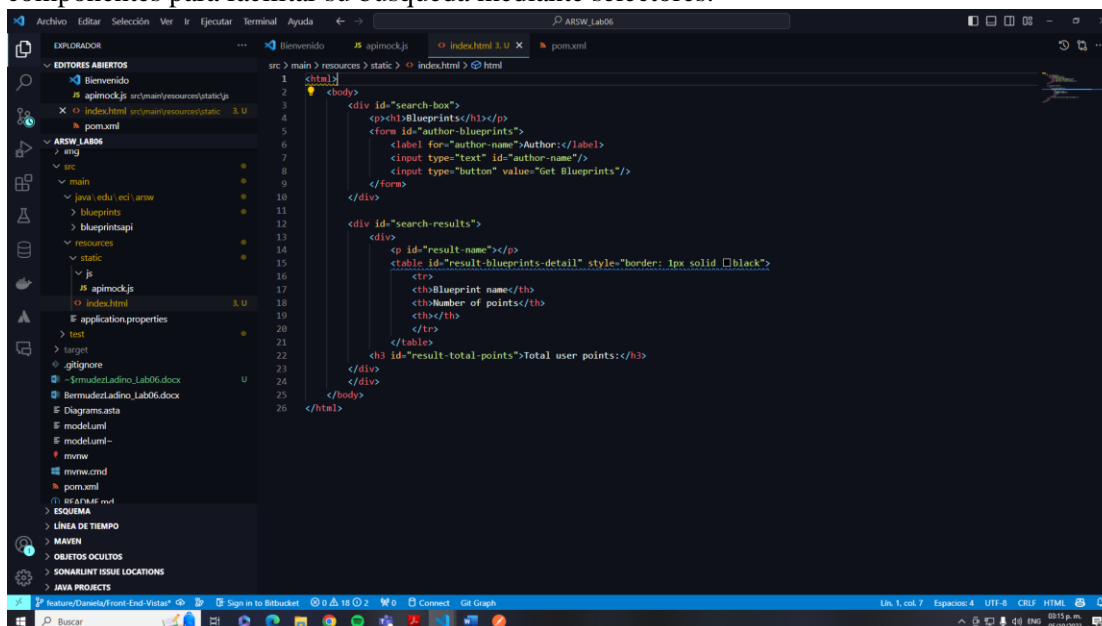
Front-End-Vistas

1. Cree el directorio donde residirá la aplicación JavaScript. Como se está usando SpringBoot, la ruta para poner en el mismo contenido estático (páginas Web estáticas, aplicaciones HTML5/JS, etc) es:

```
src/main/resources/static
```



2. Cree, en el directorio anterior, la página `index.html`, sólo con lo básico: título, campo para la captura del autor, botón de 'Get blueprints', campo donde se mostrará el nombre del autor seleccionado, [la tabla HTML](#) donde se mostrará el listado de planos (con sólo los encabezados), y un campo donde se mostrará el total de puntos de los planos del autor. Recuerde asociarles identificadores a dichos componentes para facilitar su búsqueda mediante selectores.



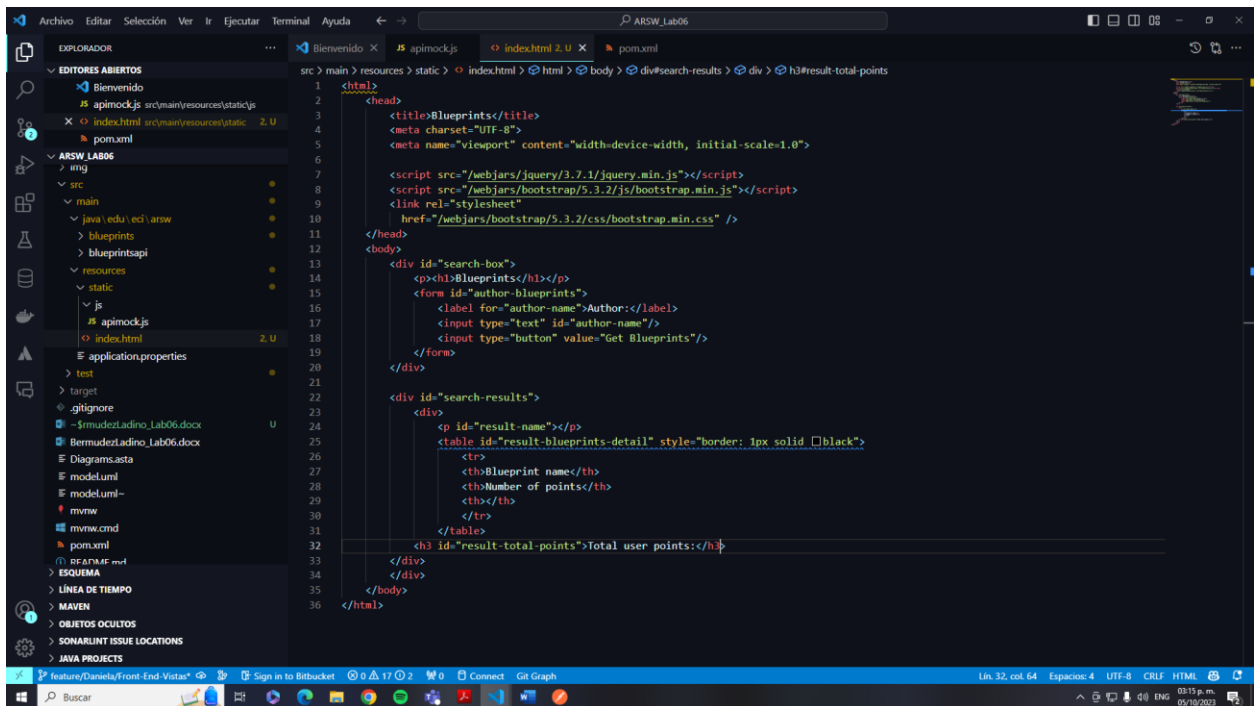
- En el elemento `<head>` de la página, agregue las referencias a las librerías de jQuery, Bootstrap y a la hoja de estilos de Bootstrap.

```
<head>

  <title>Blueprints</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <script src="/webjars/jquery/jquery.min.js"></script>
  <script src="/webjars/bootstrap/3.3.7/js/bootstrap.min.js"></script>
  <link rel="stylesheet"
        href="/webjars/bootstrap/3.3.7/css/bootstrap.min.css" />
</head>
```

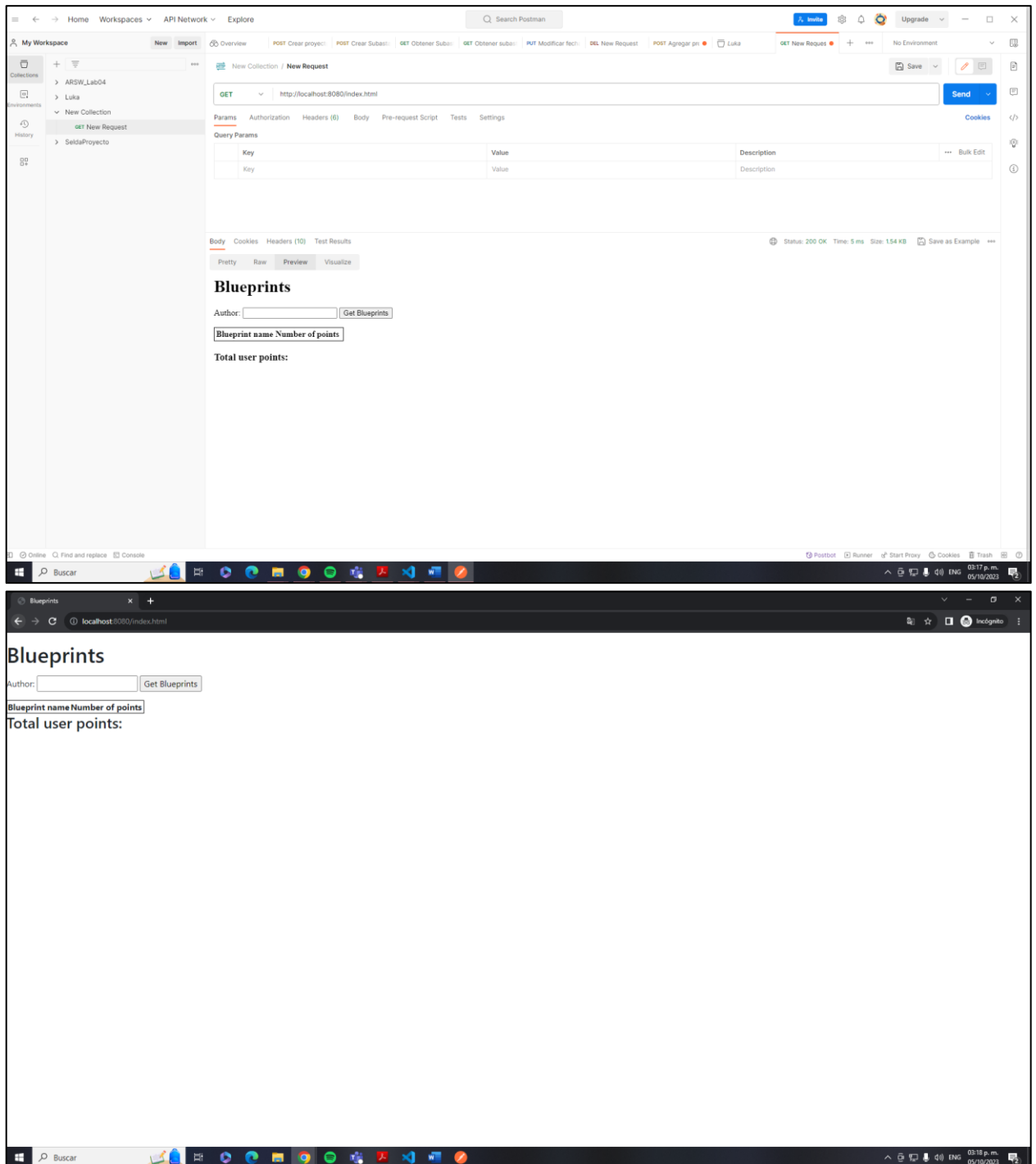
En las referencias de las librerías actualizamos las versiones con las que tenemos en el pom



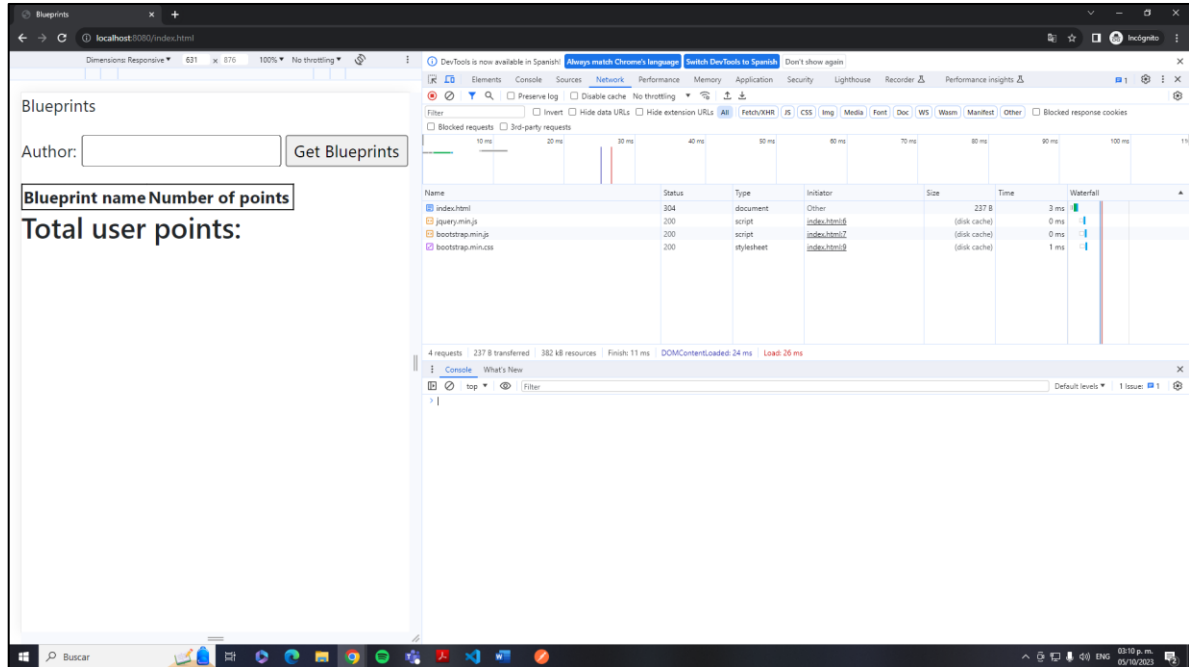
4. Suba la aplicación (mvn spring-boot:run), y rectifique:

i. Que la página sea accesible desde:

<http://localhost:8080/index.html>

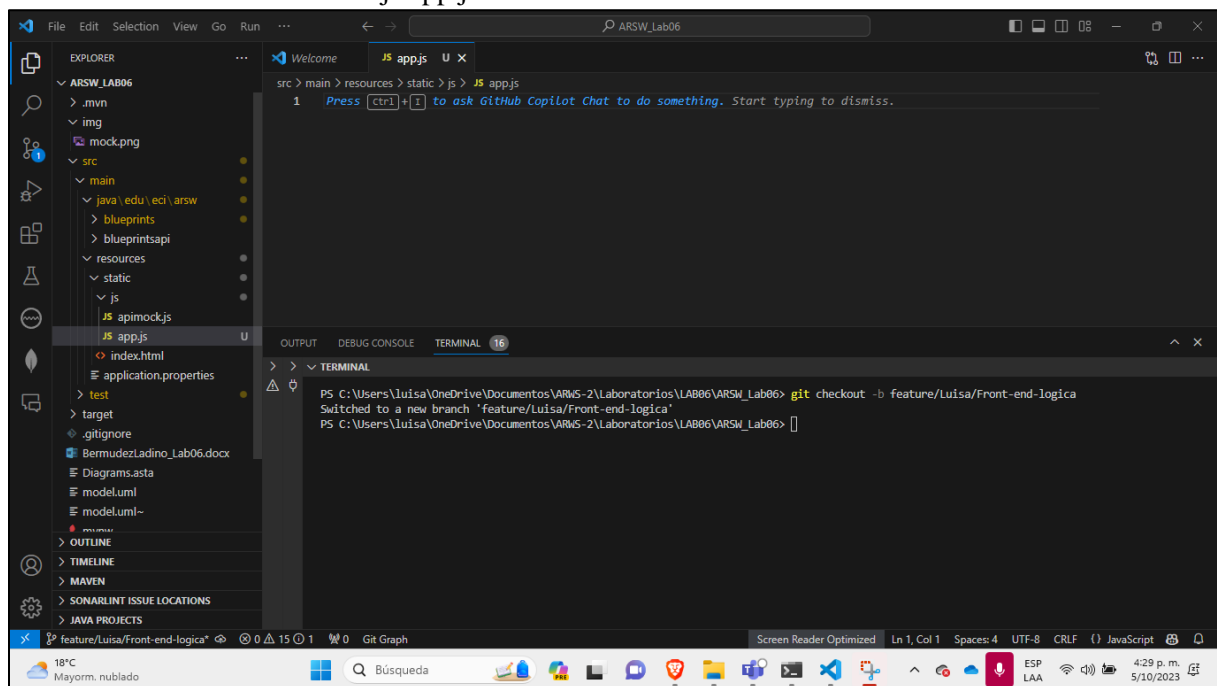


- ii. Al abrir la consola de desarrollador del navegador, NO deben aparecer mensajes de error 404 (es decir, que las librerías de JavaScript se cargaron correctamente).

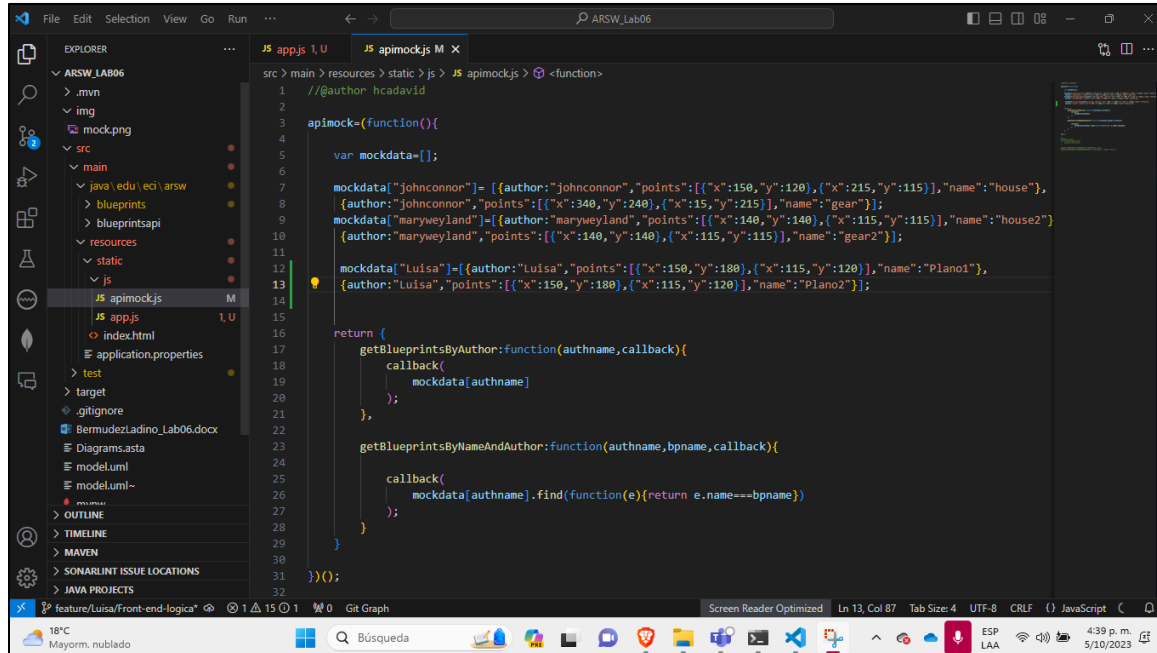


Front-End-Lógica

1. Ahora, va a crear un Módulo JavaScript que, a manera de controlador, mantenga los estados y ofrezca las operaciones requeridas por la vista. Para esto tenga en cuenta el [patrón Módulo de JavaScript](#), y cree un módulo en la ruta static/js/app.js.



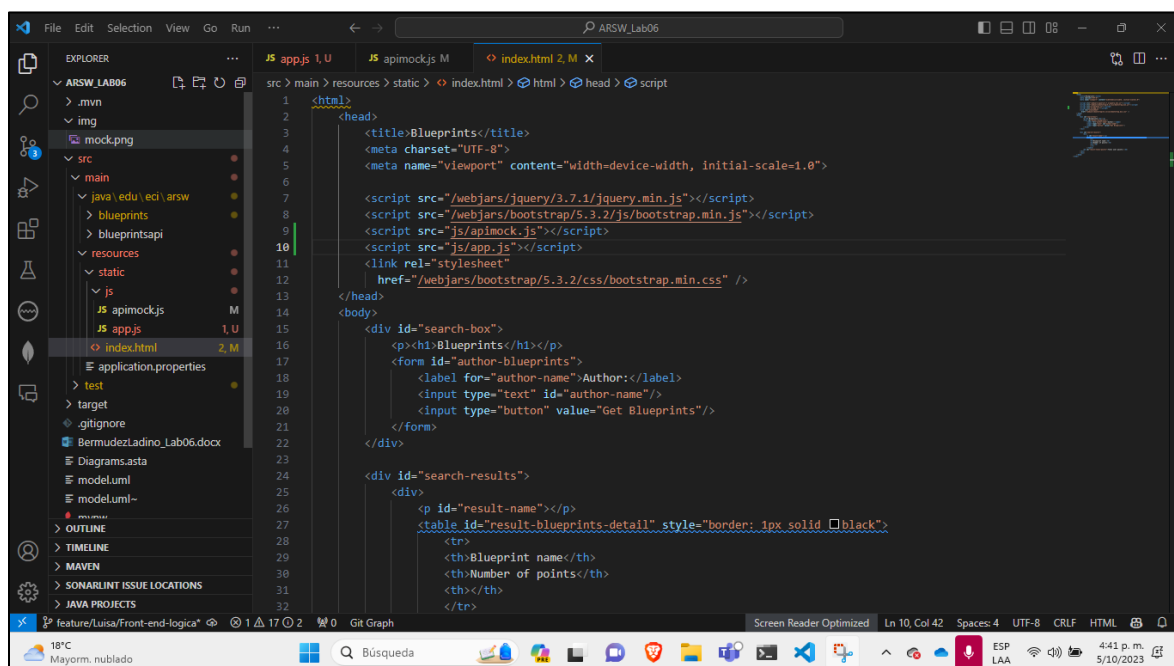
2. Copie el módulo provisto (apimock.js) en la misma ruta del módulo antes creado. En éste agréguele más planos (con más puntos) a los autores 'quemados' en el código.



```
1 // @author hcadavid
2
3 apimock=(function(){
4
5     var mockdata=[];
6
7     mockdata["johnconnor"]=[{author:"johnconnor", "points":[{"x":150,"y":120}, {"x":215,"y":115}], "name":"house"},
8     {author:"johnconnor", "points":[{"x":340,"y":240}, {"x":15,"y":215}], "name":"gear"}];
9     mockdata["maryweyland"]=[{author:"maryweyland", "points":[{"x":140,"y":140}, {"x":115,"y":115}], "name":"house2"},
10    {author:"maryweyland", "points":[{"x":140,"y":140}, {"x":115,"y":115}], "name":"gear2"}];
11
12    mockdata["Luisa"]=[{author:"Luisa", "points":[{"x":150,"y":180}, {"x":115,"y":120}], "name":"Plano1"},
13    {author:"Luisa", "points":[{"x":150,"y":180}, {"x":115,"y":120}], "name":"Plano2"}];
14
15    return {
16        getBlueprintsByAuthor:function(authname, callback){
17            callback(
18                mockdata[authname]
19            );
20        },
21        getBlueprintsByNameAndAuthor:function(authname, bpname, callback){
22            callback(
23                mockdata[authname].find(function(e){return e.name===bpname})
24            );
25        }
26    }
27 })();
```

3. Agregue la importación de los dos nuevos módulos a la página HTML (después de las importaciones de las librerías de jQuery y Bootstrap):

```
<script src="js/apimock.js"></script>
<script src="js/app.js"></script>
```

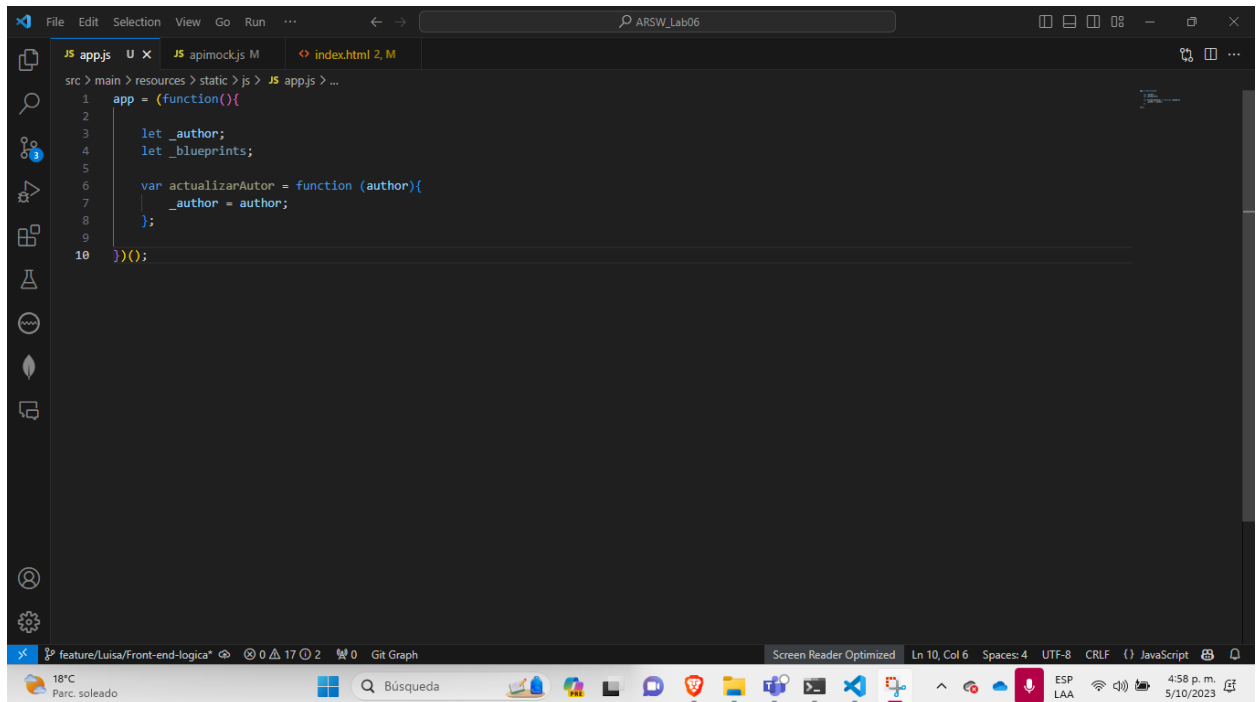


```
1 <html>
2 <head>
3 <title>Blueprints</title>
4 <meta charset="UTF-8">
5 <meta name="viewport" content="width=device-width, initial-scale=1.0">
6
7 <script src="/webjars/jquery/3.7.1/jquery.min.js"></script>
8 <script src="/webjars/bootstrap/5.3.2/js/bootstrap.min.js"></script>
9 <script src="js/apimock.js"></script>
10 <script src="js/app.js"></script>
11 <link rel="stylesheet"
12 href="/webjars/bootstrap/5.3.2/css/bootstrap.min.css" />
13 </head>
14 <body>
15 <div id="search-box">
16 <p><h1>Blueprints</h1><p>
17 <form id="author-blueprints">
18 <label for="author-name">Author:</label>
19 <input type="text" id="author-name"/>
20 <input type="button" value="Get Blueprints"/>
21 </form>
22 </div>
23
24 <div id="search-results">
25 <div>
26 <p id="result-name"><p>
27 <table id="result-blueprints-detail" style="border: 1px solid black">
28 <tr>
29 <th>Blueprint name</th>
30 <th>Number of points</th>
31 <th></th>
32 </tr>
```


4. Haga que el módulo antes creado mantenga de forma privada:

- El nombre del autor seleccionado.
- El listado de nombre y tamaño de los planos del autor seleccionado. Es decir, una lista objetos, donde cada objeto tendrá dos propiedades: nombre de plano, y número de puntos del plano.

Junto con una operación pública que permita cambiar el nombre del autor actualmente seleccionado.



```
1 app = (function(){
2
3   let _author;
4   let _blueprints;
5
6   var actualizarAutor = function (author){
7     _author = author;
8   };
9
10 })();
```

5. Agregue al módulo 'app.js' una operación pública que permita actualizar el listado de los planos, a partir del nombre de su autor (dado como parámetro). Para hacer esto, dicha operación debe invocar la operación 'getBlueprintsByAuthor' del módulo 'apimock' provisto, enviándole como callback una función que:

- Tome el listado de los planos, y le aplique una función 'map' que convierta sus elementos a objetos con sólo el nombre y el número de puntos.
- Sobre el listado resultante, haga otro 'map', que tome cada uno de estos elementos, y a través de jQuery agregue un elemento <tr> (con los respectivos <td>) a la tabla creada en el punto 4. Tenga en cuenta los [selectores de jQuery](#) y [los tutoriales disponibles en línea](#). Por ahora no agregue botones a las filas generadas.
- Sobre cualquiera de los dos listados (el original, o el transformado mediante 'map'), aplique un 'reduce' que calcule el número de puntos. Con este valor, use jQuery para actualizar el campo correspondiente dentro del DOM.

```
src > main > resources > static > js > JS apimockjs > <function> > getBlueprintsByAuthor
1  author hcadavid
2
3  mock=(function(){
4
5      var mockdata=[];
6
7      mockdata["johnconnor"]=[{author:"johnconnor","points":[{"x":150,"y":120}, {"x":215,"y":115}], "name": "house"},
8      {author:"johnconnor","points":[{"x":340,"y":240}, {"x":15,"y":215}], "name": "gear"}];
9      mockdata["maryweyland"]=[{author:"maryweyland","points":[{"x":140,"y":140}, {"x":115,"y":115}], "name": "house2"},
10     {author:"maryweyland","points":[{"x":140,"y":140}, {"x":115,"y":115}], "name": "gear2"}];
11
12     mockdata["Luisa"]=[{author:"Luisa","points":[{"x":150,"y":180}, {"x":115,"y":120}], "name": "Plano1"},
13     {author:"Luisa","points":[{"x":150,"y":180}, {"x":115,"y":120}], "name": "Plano2"}];
14
15     mockdata["Karol"]=[{author:"Karol","points":[{"x":150,"y":150}, {"x":150,"y":150}], "name": "CasaAzul"},
16     {author:"Karol","points":[{"x":110,"y":0}, {"x":150,"y":130}, {"x":150,"y":0}, {"x":110,"y":130}], "name": "CasaVerde"}];
17
18     return {
19         getBlueprintsByAuthor:function(author,callback){
20             callback(
21                 author,
22                 mockdata[author]
23             );
24         },
25     };
26 }
```

```
src > main > resources > static > js > JS appjs > <function>
4  let _blueprints;
5
6  var _refreshAuthorState = function (author, blueprintObjects) {
7
8      _author = author;
9      _blueprints = blueprintObjects.map((blueprint) => {
10         return { name: blueprint.name, puntos: blueprint.points.length }
11     });
12
13     if (author == "" || author == null) {
14         alert("Debe poner un nombre en el buscador!");
15     } else {
16         $("#result-name").text(author + "'s Blueprints:");
17     }
18     $("#result-blueprints-detail td").remove();
19     _blueprints.map((blueprint) => {
20         $("#result-blueprints-detail").append(
21             "<tr><td>" + blueprint.name + "</td>" +
22             "<td>" + blueprint.puntos + "</td>" +
23             "<td><input id=" + blueprint.name + " type = 'button' onclick='app.printBlueprint(this)' value='Open' /></td>"
24         );
25     });
26
27     let totalPuntos = _blueprints.reduce((total, currentValue) => total + currentValue.puntos, 0);
28     $("#result-total-points").text("Total user's points: " + totalPuntos);
29
30 };
31
32 var actualizarAutor = function (author){
33     _author = author;
34 };
35 }
```

```
src > main > resources > static > js > JS app.js > <function> > @_refreshAuthorState > @_blueprints.map() callback
let totalPuntos = _blueprints.reduce((total, currentValue) => total + currentValue.puntos, 0);
$("#result-total-points").text("Total user's points: " + totalPuntos);

};

var actualizarAutor = function (author){
  _author = author;
};

var actualizarBlueprint = function(author){
  _author = author;
  apimock.getBlueprintsByAuthor(author, _refreshAuthorState);
};

var innerMockModule = {
  getAuthorBlueprints: function () {
    let author = $("#author-name").val();
    actualizarBlueprint(author);
  }
};
return innerMockModule;
})();
```

6. Asocie la operación antes creada (la de app.js) al evento 'on-click' del botón de consulta de la página.

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">

<script src="/webjars/jquery/3.7.1/jquery.min.js"></script>
<script src="/webjars/bootstrap/5.3.2/js/bootstrap.min.js"></script>
<script src="/js/apimock.js"></script>
<script src="/js/app.js"></script>
<link rel="stylesheet" href="/webjars/bootstrap/5.3.2/css/bootstrap.min.css" />

</head>
<body>
  <div id="search-box">
    <p><h1>Blueprints</h1></p>
    <form id="author-blueprints">
      <label for="author-name">Author:</label>
      <input type="text" id="author-name"/>
      <input type="button" onclick="app.getAuthorBlueprints()" value="Get Blueprints"/>
    </form>
  </div>

  <div id="search-results">
    <div>
      <p id="result-name"></p>
      <table id="result-blueprints-detail" style="border: 1px solid black">
        <tr>
          <th>Blueprint name</th>
          <th>Number of points</th>
        </tr>
      </table>
    </div>
    <h3 id="result-total-points">Total user points:</h3>
  </div>
```

7. Verifique el funcionamiento de la aplicación. Inicie el servidor, abra la aplicación HTML5/JavaScript, y rectifique que, al ingresar un usuario existente, se cargue el listado del mismo.

The screenshot shows a web browser window with the title 'Blueprints'. The address bar shows 'localhost:8080/index.html'. The page content includes a form with 'Author: Luisa' and a 'Get Blueprints' button. Below the form, it says 'Luisa's Blueprints:' followed by a table:

Blueprint name	Number of points
Plano1	2
Plano2	2

Below the table, it says 'Total user's points: 4'. The browser's taskbar at the bottom shows the Windows logo, a search bar with 'Búsqueda', and various application icons. The system tray shows the date and time as '5:48 p. m. 5/10/2023'.

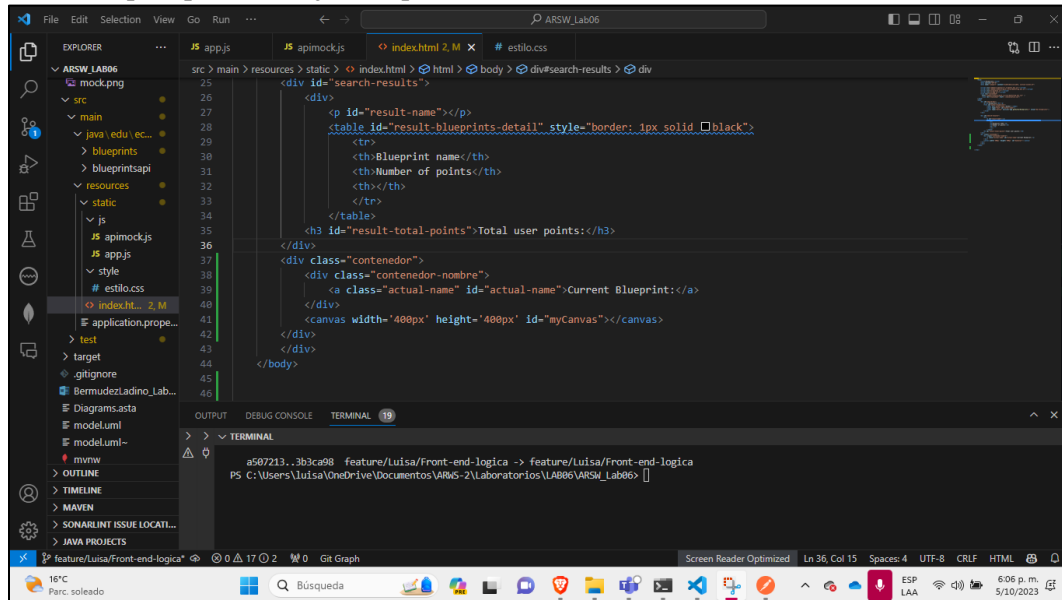
The screenshot shows the same web browser window with the title 'Blueprints'. The address bar shows 'localhost:8080/index.html'. The page content includes a form with 'Author: Karol' and a 'Get Blueprints' button. Below the form, it says 'Karol's Blueprints:' followed by a table:

Blueprint name	Number of points
CasaAzul	2
CasaVerde	4

Below the table, it says 'Total user's points: 6'. The browser's taskbar at the bottom shows the Windows logo, a search bar with 'Búsqueda', and various application icons. The system tray shows the date and time as '5:48 p. m. 5/10/2023'.

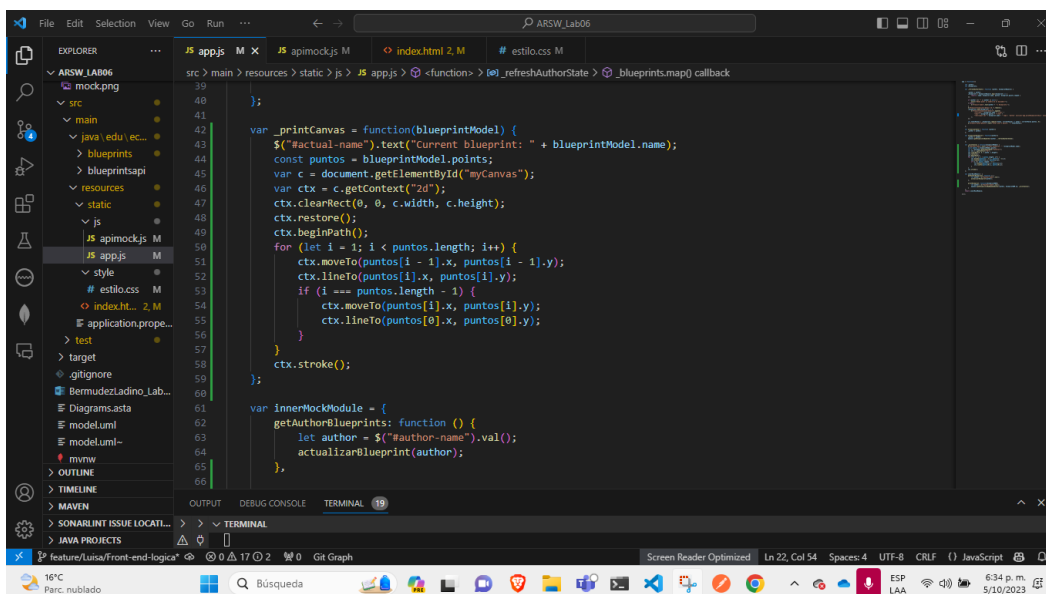
Para la próxima semana

8. A la página, agregue un [elemento de tipo Canvas](#), con su respectivo identificador. Haga que sus dimensiones no sean demasiado grandes para dejar espacio para los otros componentes, pero lo suficiente para poder 'dibujar' los planos.



9. Al módulo app.js agregue una operación que, dado el nombre de un autor, y el nombre de uno de sus planos dados como parámetros, haciendo uso del método getBlueprintsByNameAndAuthor de apimock.js y de una función callback:

- Consulte los puntos del plano correspondiente, y con los mismos dibuje consecutivamente segmentos de recta, haciendo uso [de los elementos HTML5 \(Canvas, 2DContext, etc\) disponibles](#)* Actualice con jQuery el campo donde se muestra el nombre del plano que se está dibujando (si dicho campo no existe, agréguelo al DOM).



The screenshot shows a VS Code editor window with the following details:

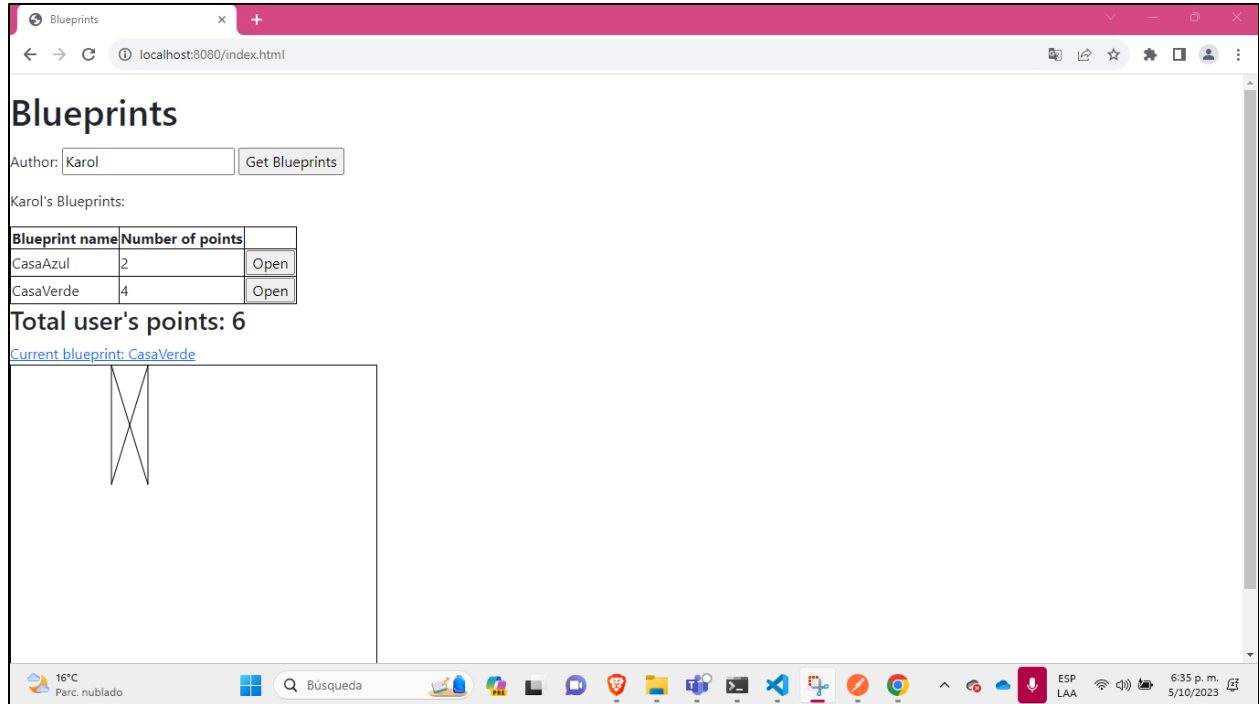
- Explorer:** A file tree on the left showing a project structure with folders like `src`, `main`, `resources`, and `static`. The `src` folder is expanded, showing `main`, `resources`, and `static`.
- Editor:** The main area displays a JavaScript file named `app.js`. The code includes a `for` loop that iterates over an array of points (`puntos`) and draws lines on a canvas context (`ctx`). It also defines a `printBlueprint` function that interacts with a mock API (`apimock`) to retrieve blueprints by name and author.
- Terminal:** The bottom panel shows the `TERMINAL` tab, which is currently empty.
- Status Bar:** The bottom status bar indicates the file is `feature/Luisa/Front-end-logica`, with a line count of 22 and a column count of 54.

10. Verifique que la aplicación ahora, además de mostrar el listado de los planos de un autor, permita seleccionar uno de éstos y graficarlo. Para esto, haga que en las filas generadas para el punto 5 incluyan en la última columna un botón con su evento de clic asociado a la operación hecha anteriormente (enviando como parámetro los nombres correspondientes).

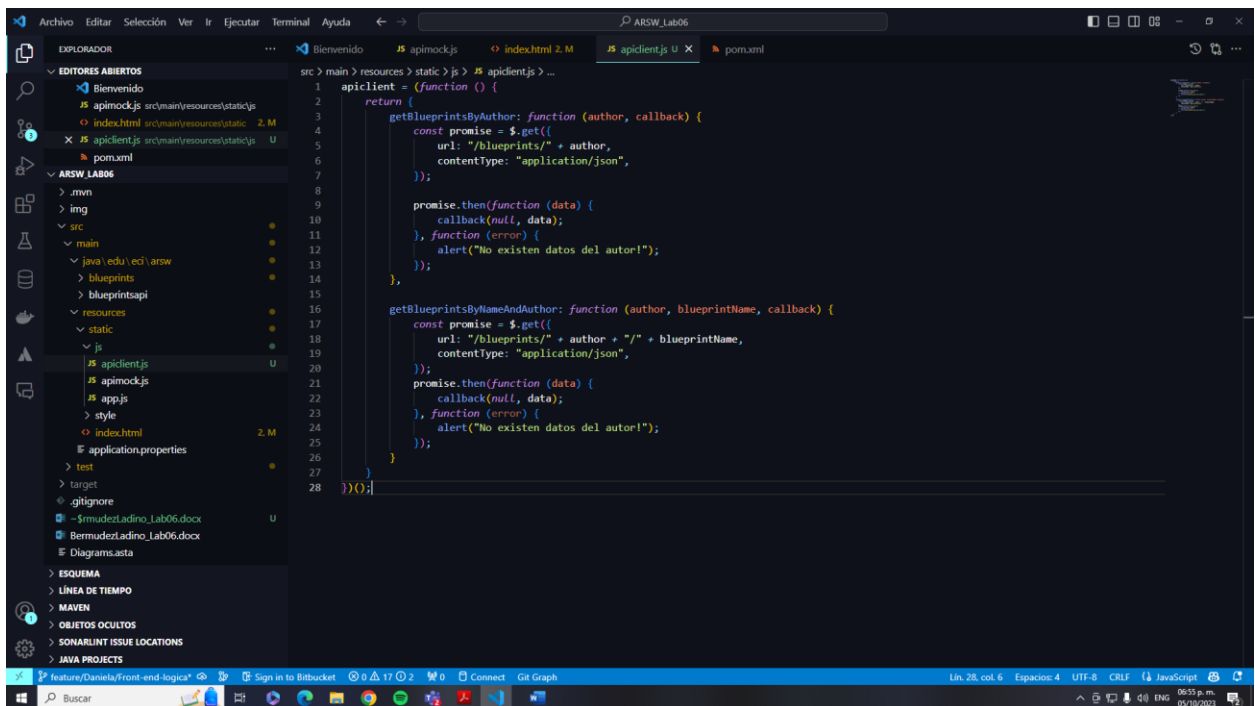
The screenshot shows a VS Code editor window with the following details:

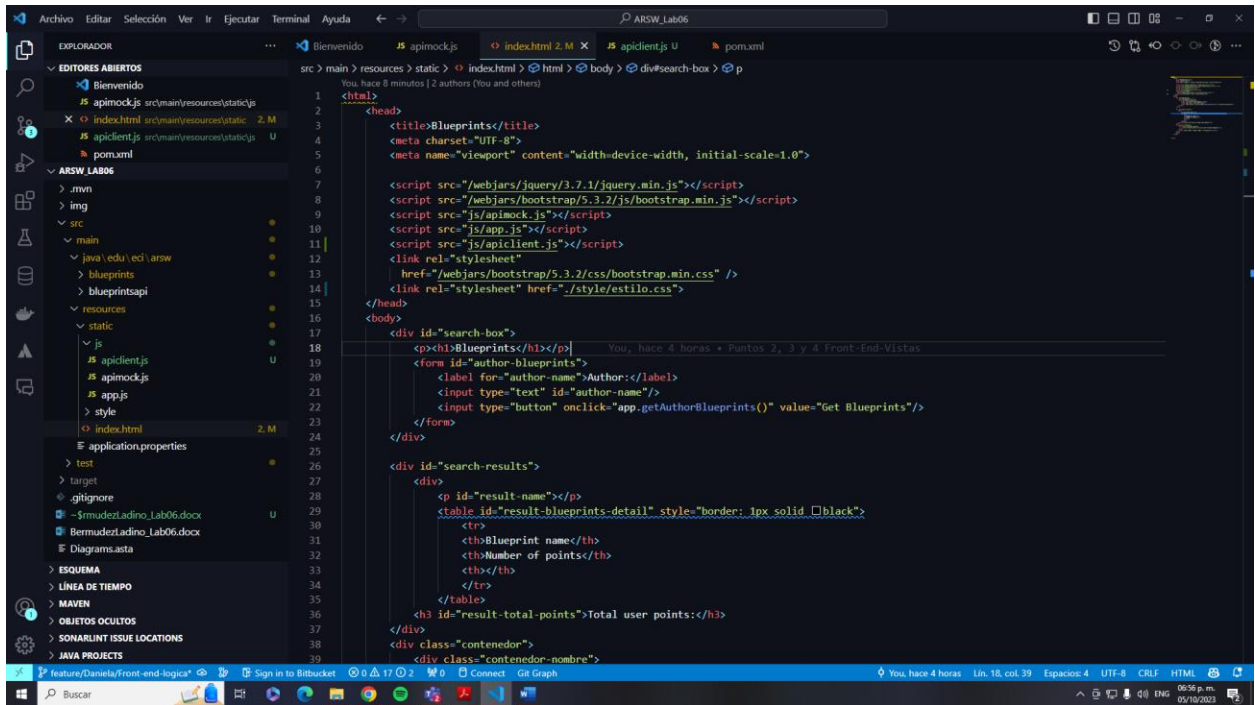
- Explorer:** A file tree on the left showing a project structure with folders like `src`, `main`, `resources`, and `static`. The `src` folder is expanded, showing `main`, `resources`, and `static`.
- Editor:** The main area displays a JavaScript file named `app.js`. The code includes a `for` loop that iterates over an array of points (`puntos`) and draws lines on a canvas context (`ctx`). It also defines a `printBlueprint` function that interacts with a mock API (`apimock`) to retrieve blueprints by name and author.
- Terminal:** The bottom panel shows the `TERMINAL` tab, which is currently empty.
- Status Bar:** The bottom status bar indicates the file is `feature/Luisa/Front-end-logica`, with a line count of 22 and a column count of 54.

11. Verifique que la aplicación ahora permita: consultar los planos de un auto y graficar aquel que se seleccione.

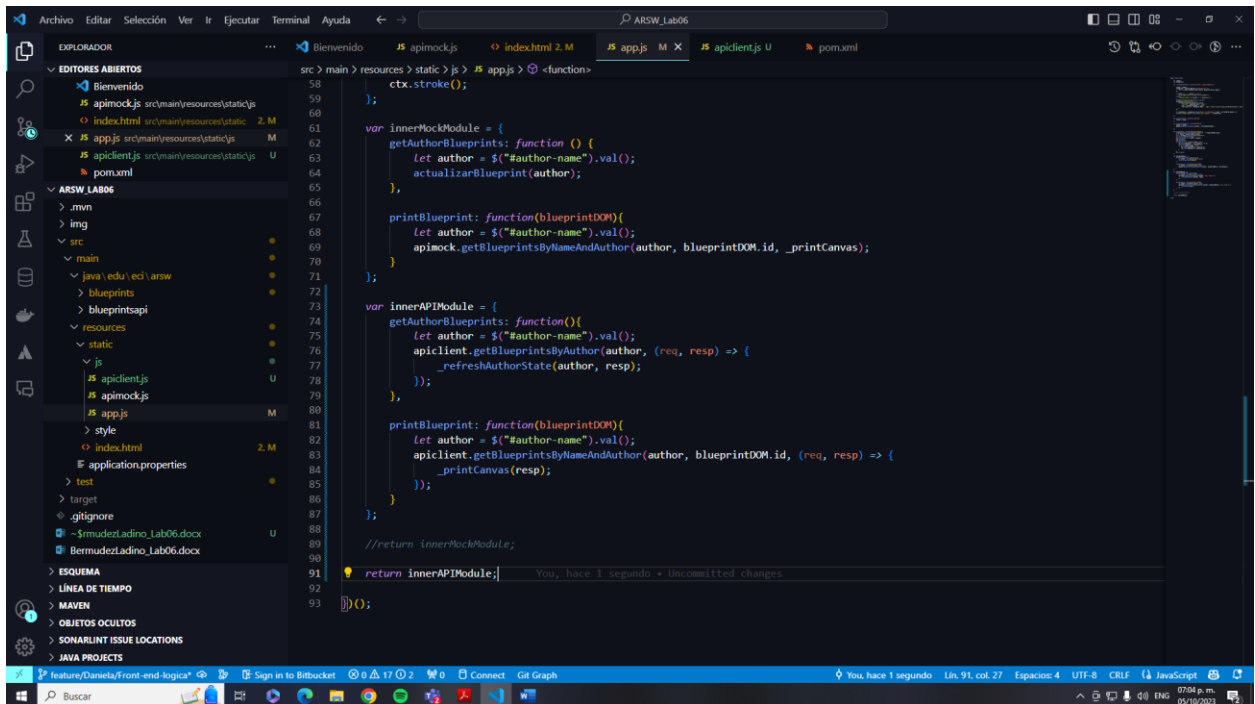


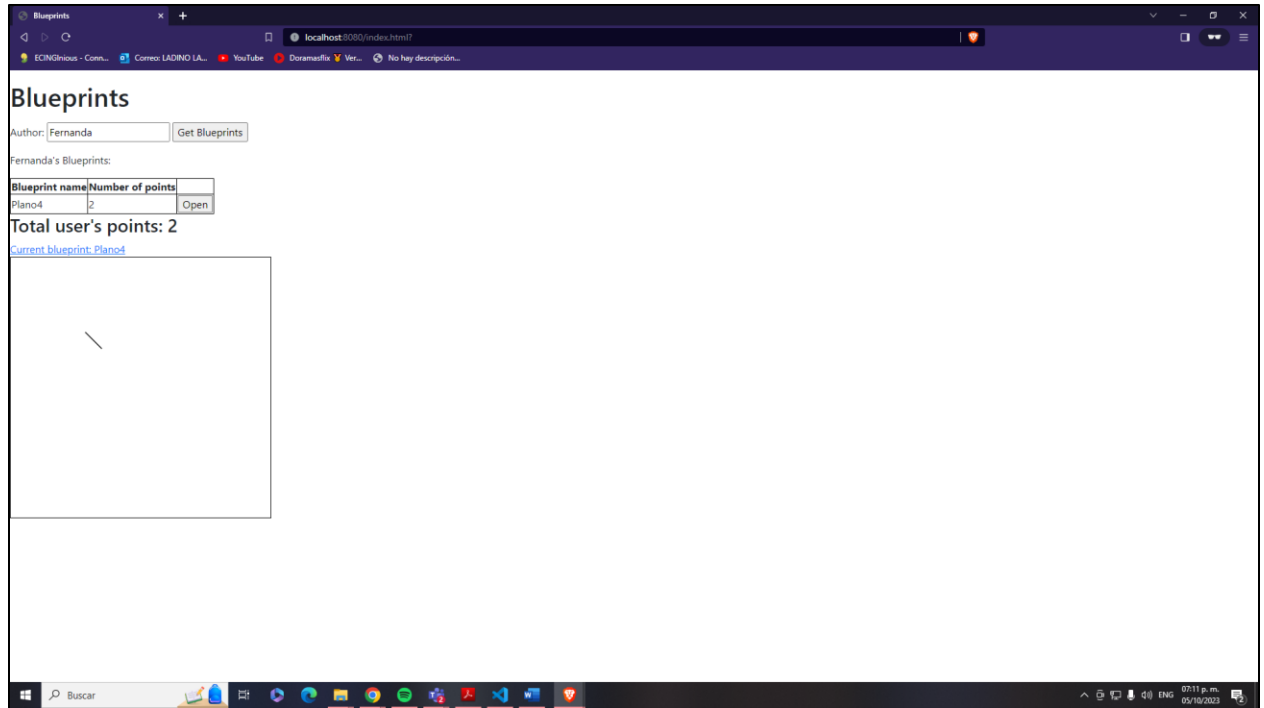
12. Una vez funcione la aplicación (sólo front-end), haga un módulo (llámelo 'apiclient') que tenga las mismas operaciones del 'apimock', pero que para las mismas use datos reales consultados del API REST. Para lo anterior revise [cómo hacer peticiones GET con jQuery](#), y cómo se maneja el esquema de callbacks en este contexto.





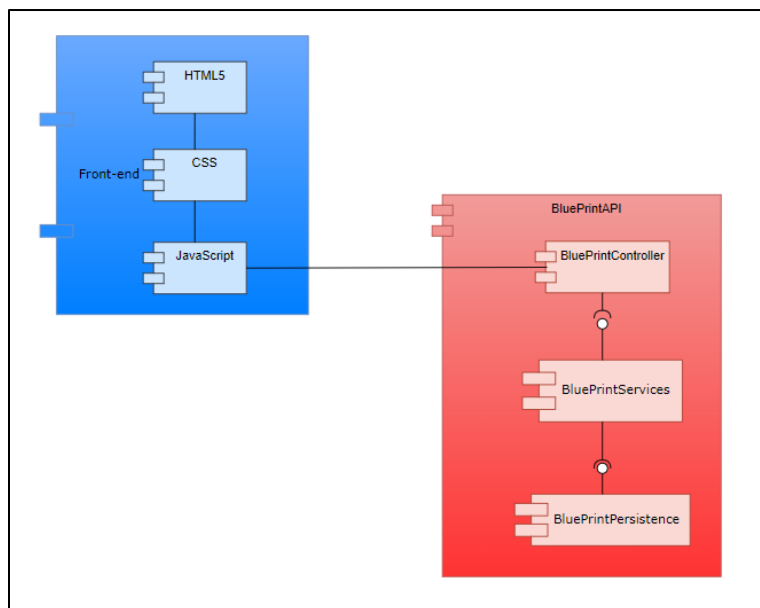
13. Modifique el código de app.js de manera que sea posible cambiar entre el 'apimock' y el 'apiclient' con sólo una línea de código.





14. Revise la [documentación y ejemplos de los estilos de Bootstrap](#) (ya incluidos en el ejercicio), agregue los elementos necesarios a la página para que sea más vistosa, y más cercana al mock dado al inicio del enunciado.

Diagrama de Componentes



III. Conclusiones

- El laboratorio permitió desarrollar un cliente "grueso" que utiliza tecnologías web modernas como HTML5, JavaScript y CSS3 para interactuar con un servicio de consulta de planos de autor.
- El laboratorio se compone de dos partes una es el back-end que implementa el servicio de consulta de planos, y la otra el front-end que interactúa con el back-end para mostrar la información a los usuarios. La parte front-end se desarrolló utilizando las librerías jQuery y Bootstrap.