CONSTRUCIÓN DE UN CLIENTE 'GRUESO' CON UN API REST, HTML5, JAVASCRIPT Y CSS3. PARTE I.

AUTORES:

YORKS GOMEZ – CESAR VÁSQUEZ

PROFESOR:

JAVIER IVAN TOQUICA

ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

ARQUITECTURAS DE SOTFWARE - GRUPO # 1

INGENIERÍA DE SISTEMAS

BOGOTÁ D.C.

2023

INTRODUCCIÓN

En este laboratorio le daremos al cliente una vista generosa, ágil y fácil de entender y usar para consultar planos de un usuario. Con las herramientas API REST, HTML5, JAVASCRIPT Y CSS3 para el front y teniendo como base del back el laboratorio anterior. En la consulta aparecerán los planos del usuario dado en un campo de texto en el formulario. Por ahora, si la consulta genera un error, sencillamente no se mostrará nada. Al hacer una consulta exitosa, se mostrará un mensaje que incluya el nombre del autor, y una tabla con: el nombre de cada plano de autor, el número de puntos de este, y un botón para abrirlo. Al final, se mostrará el total de puntos de todos los planos. Al seleccionar uno de los planos, se debe mostrar el dibujo de este. Por ahora, el dibujo será simplemente una secuencia de segmentos de recta realizada en el mismo orden en el que vengan los puntos.

Usaremos herramientas como JQuery, Ajax, API’s, entre otras para la conexión correcta entre el Front-end y el Back-end.

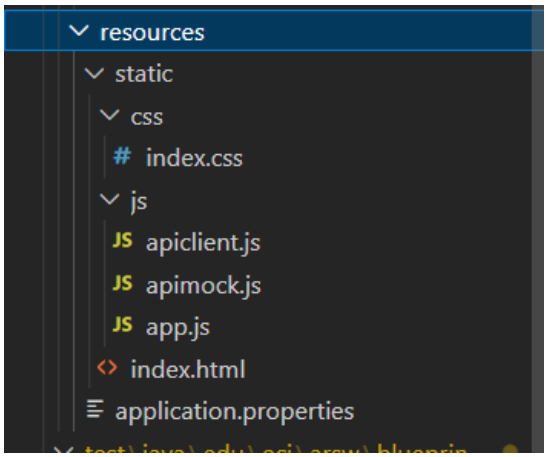
DESARROLLO

Incluimos dentro de las dependencias de Maven los 'webjars' de jQuery y Bootstrap (lo cual nos permite tener localmente dichas librerías de JavaScript al momento de construír 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>
</dependencies>
```

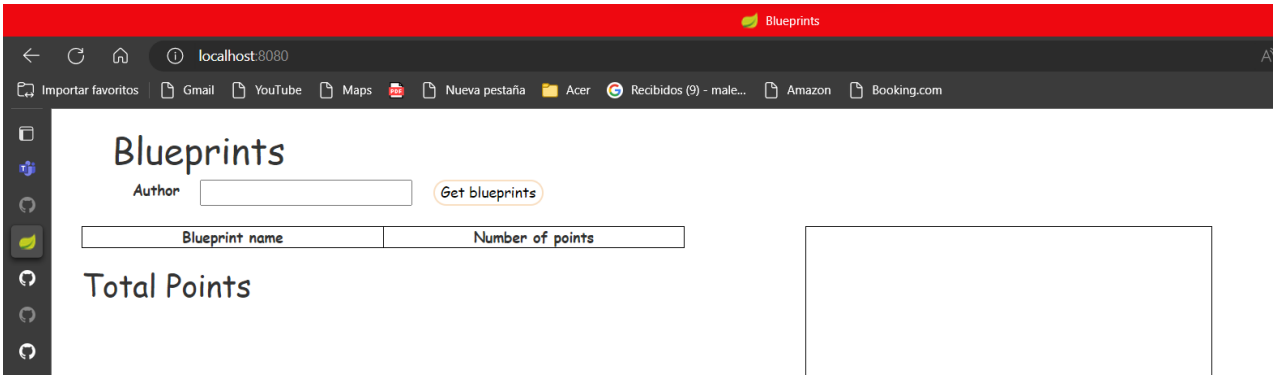
Front-End – Vistas

Creamos 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



Creamos, 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.

```
    <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" />

    <link rel="stylesheet" href="css/index.css" />
</head>
<body>
    <h1 class="titulo">Blueprints</h1>
    <div class="autor">
        <span>Author</span><input id="author">
        <button onclick="app.getNameAuthorBlueprints()">Get blueprints</button>
    </div>
    <div class="grid">
        <div class="tabla">
            <h2 id="name"></h2>
            <table id="table">
                <thead>
                <tr>
                    <th>Blueprint name</th>
                    <th>Number of points</th>
                    <th>Draw</th>
                </tr>
                </thead>
                <tbody>
                </tbody>
            </table>
            <h2 id="points">Total Points</h2>
        </div>
        <div class="canva">
            <h2 id="nameblu"></h2>
            <canvas id="myCanvas" width="400" height="400" style="border:1px solid #000000;"></canvas>
        </div>
    </div>
```



Front-End – Lógica

Añadimos más planos a los autores johnconnor y maryweyland

```
apimock=(function(){

    var mockdata=[];

    mockdata["johnconnor"]= [
        {author:"johnconnor","points":[{"x":150,"y":120},{"x":215,"y":115}],"name":"house"},
        {author:"johnconnor","points":[{"x":340,"y":240},{"x":15,"y":215}],"name":"gear"},
        {author:"johnconnor","points":[{"x":34,"y":76},{"x":190,"y":71}],"name":"tree"},
        {author:"johnconnor","points":[{"x":370,"y":210},{"x":152,"y":212}],"name":"car"}];

    mockdata["maryweyland"]=[
        {author:"maryweyland","points":[{"x":140,"y":140},{"x":115,"y":115}],"name":"house2"},
        {author:"maryweyland","points":[{"x":140,"y":140},{"x":115,"y":115}],"name":"gear2"},
        {author:"maryweyland","points":[{"x":120,"y":120},{"x":115,"y":125}],"name":"tree2"},
        {author:"maryweyland","points":[{"x":130,"y":111},{"x":125,"y":135}],"name":"car2"}];
```

Agregamos la importación de los dos nuevos módulos de la pagina html

```
    </div>
<!--Importamos los Archivos -->
<script src="js/apimock.js"></script>
<script src="js/app.js"></script>
```

Agregamos 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 esta función:

```
function getNameAuthorBlueprints() {
    author = $("#author").val();
    if (author === "") {
        alert("Debe ingresar un nombre");
    } else {
        apimock.getBlueprintsByAuthor(author,authorData);
    }
}
```

Tomamos el listado de los planos, y le aplicamos una función 'map' que convierta sus elementos a objetos con sólo el nombre y el número de puntos.

```
var authorData = function( data) {
    $("#table tbody").empty();
    if (data === undefined) {
        alert("No existe el autor");
        $("#name").empty();
        $("#points").text("Total Points");
        $("#nameblu").empty();
    } else {
        getName();
        const datanew = data.map((elemento) => {
            return {
                name: elemento.name,
                puntos: elemento.points.length
            }
        });
```

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 respectvos <td>) a la tabla creada anteriormente.

```
datanew.map((elementos) => {
    $("#table > tbody:last").append($("<tr><td>" + elementos.name + "</td><td>" + elementos.puntos.toString() +
        "</td><td>" + "<button  id=" + elementos.name + " onclick=app.getBlueprintByAuthorAndName(this)>open</button>" + "</td>"));
});
```

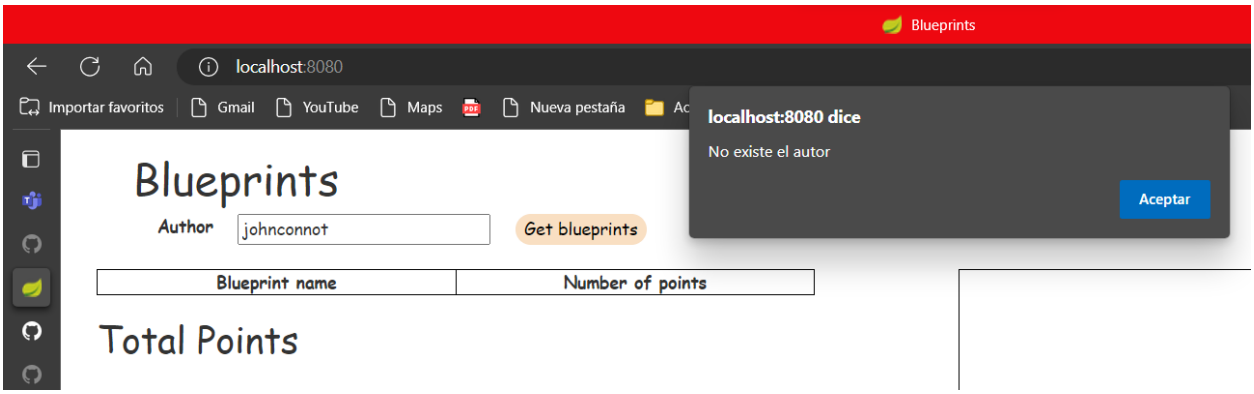
Sobre cualquiera de los dos listados (el original, o el transformado mediante 'map'), aplicamos un 'reduce' que calcule el número de puntos. Con este valor, use jQuery para actualizar el campo correspondiente dentro del DOM.

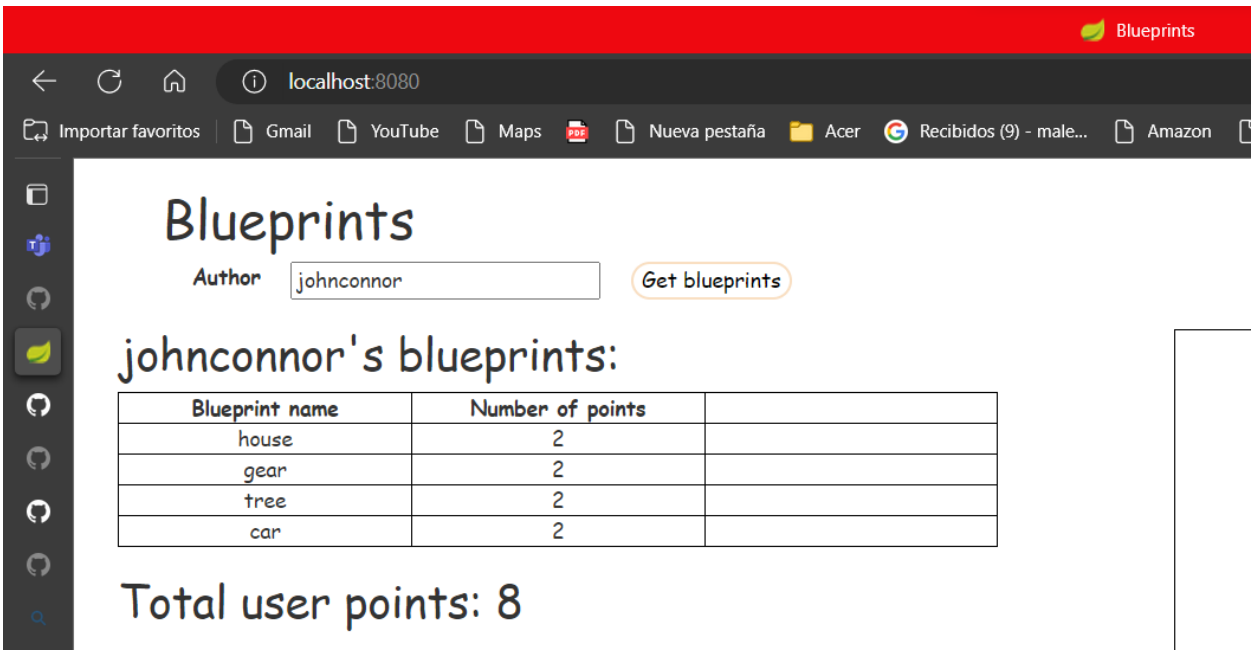
```
const totalPuntos = datanew.reduce((suma, {puntos}) => suma + puntos, 0);

$("#points").text("Total user points: " + totalPuntos);
}
```

Asociamos la operación antes creada (la de app.js) al evento 'on-click' del botón de consulta de la página. Lo realizamos en el index.html

```
<div class="autor">
    <span>Author</span><input id="author">
    <button onclick="app.getNameAuthorBlueprints()">Get blueprints</button>
</div>
```

Verificamos el funcionamiento de la aplicación



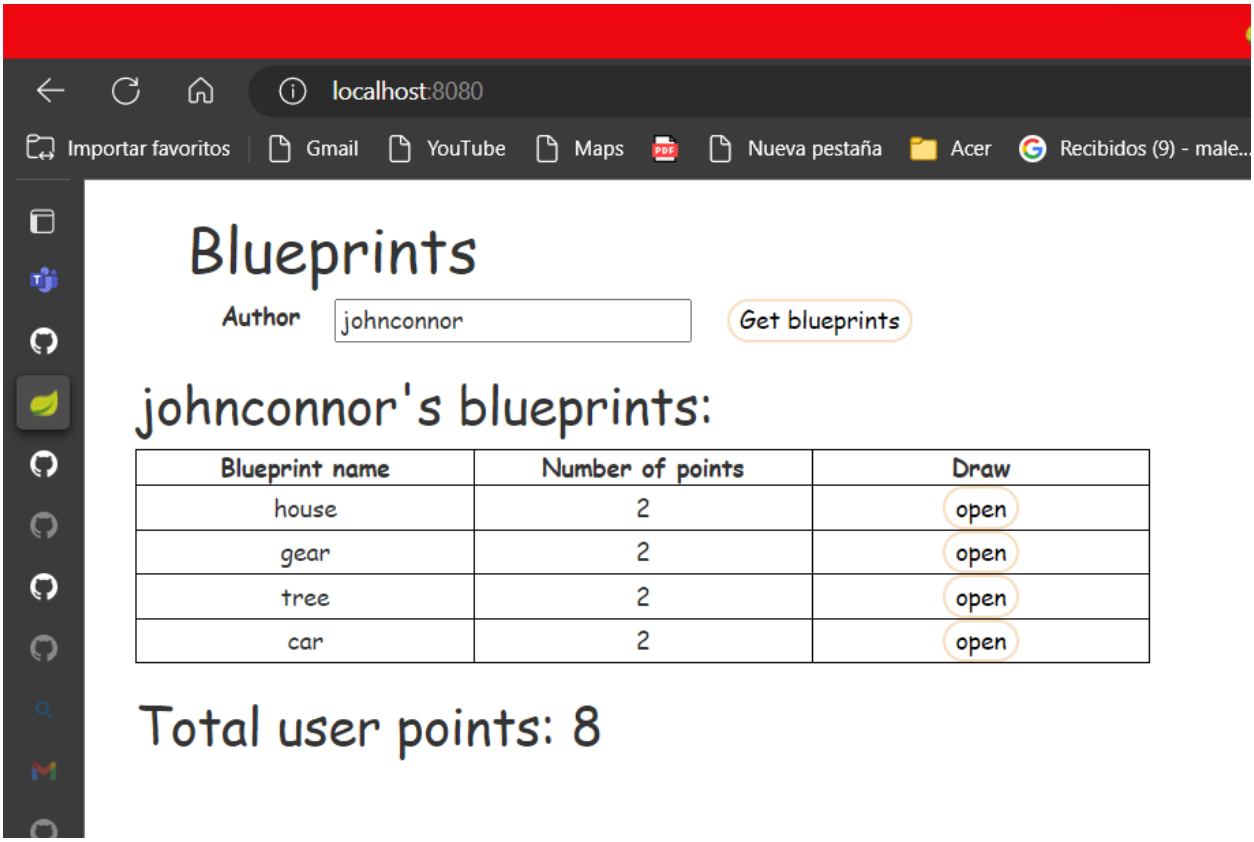


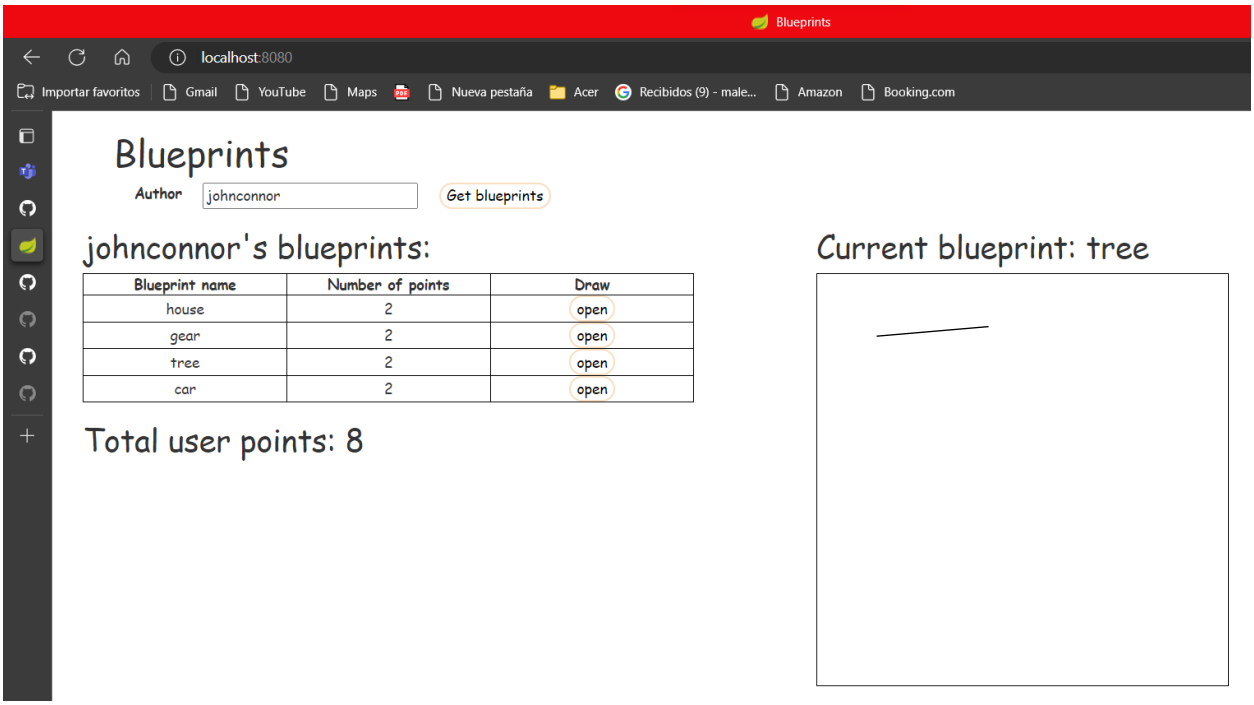
Al módulo app.js agregamos una operación que, dado el nombre de un autor, y el nombre de uno de sus planos dados como parámetros, haciendo uso del método getBlueprintsByNameAndAuthor de apimock.js y de una función callback:

- 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).

```
function getBlueprintByAuthorAndName(data) {
        author = $("#author").val();
        blueprintName = data.id;
        $("#nameblu").text("Current blueprint: " + blueprintName);
        apimock.getBlueprintByAuthorAndName(author, blueprintName, printPoints);
    }
```

Verificamos que además de mostrar el listado de los planos de un autor, permita seleccionar uno de éstos y graficarlo.






```
function printPoints(data) {
    const puntos = data.points;
    var c = document.getElementById("myCanvas");
    var ctx = c.getContext("2d");
    ctx.clearRect(0, 0, c.width, c.height);
    ctx.restore();
    ctx.beginPath();
    for (let i = 1; i < puntos.length; i++) {
        ctx.moveTo(puntos[i - 1].x, puntos[i - 1].y);
        ctx.lineTo(puntos[i].x, puntos[i].y);
        if (i === puntos.length - 1) {
            ctx.moveTo(puntos[i].x, puntos[i].y);
            ctx.lineTo(puntos[0].x, puntos[0].y);
        }
    }
    ctx.stroke();
}
```

```
return{
    getBlueprintByAuthorAndName:getBlueprintByAuthorAndName,
    getNameAuthorBlueprints: getNameAuthorBlueprints
}
```

Una vez funciona 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.

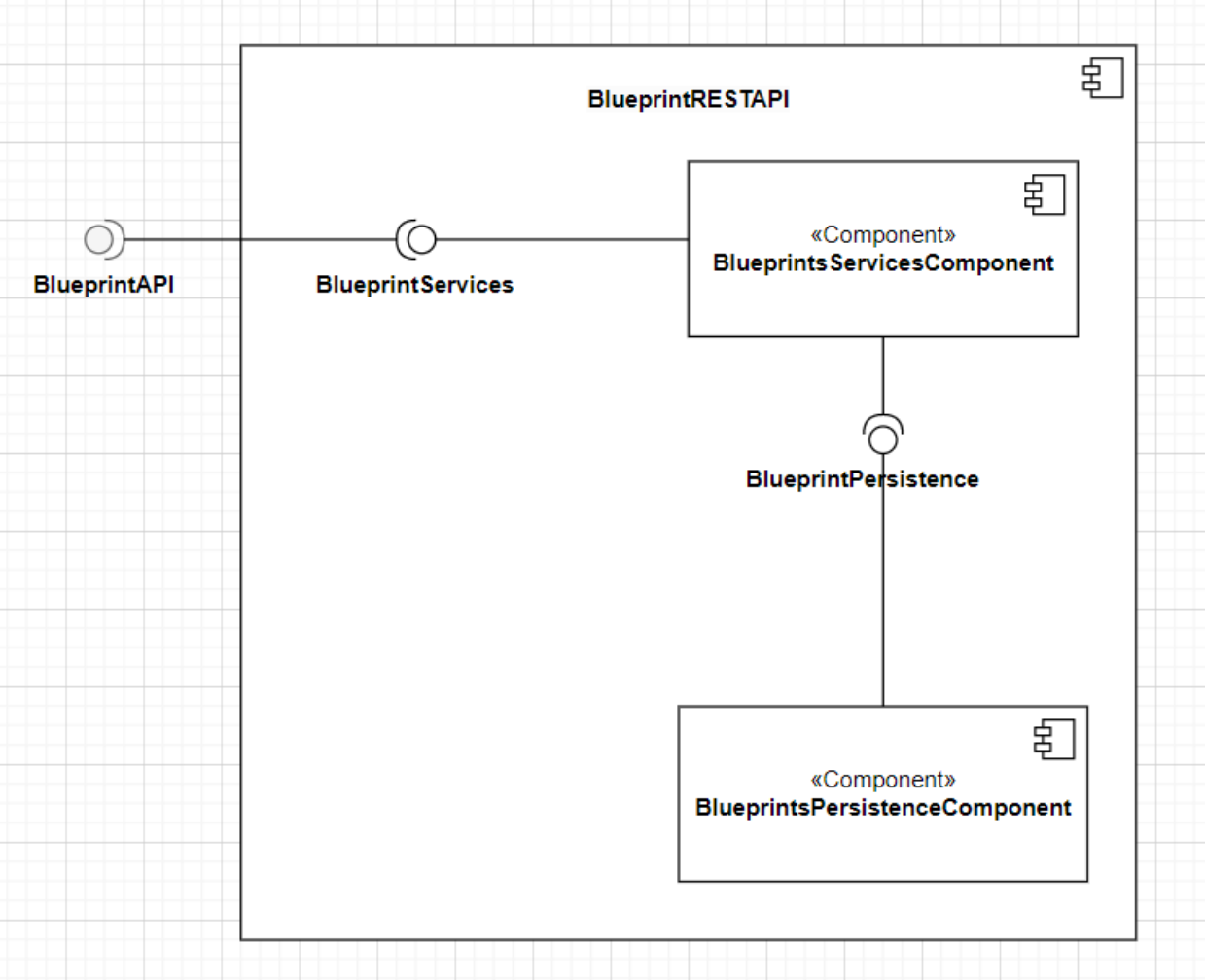
```
var apiclient = (function(){
    return {
        getBlueprintsByAuthor: function(author, callback){
        callback(
            JSON.parse($.ajax({type: 'GET', url: 'blueprints/' + author, async: false}).responseText)
        )},

        getBlueprintByAuthorAndName: function(author, bpname, callback){
        var link = author + "/" + bpname;
        callback(
            JSON.parse($.ajax({type: 'GET', url: 'blueprints/' + link, async: false}).responseText)
        )}
    }
})();
```

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

```
src > main > resources > static > js > JS app.js > [e] app >
1    var apimock = apimock;
2
```

Desde el inicio del front de la aplicación agregamos el archivo .css para dar estilo a la pagina y que fuera agradable para el usuario.

DIAGRAMA DE COMPONENTES



CONCLUSIONES

- Aprendimos a realizar la conexión correcta entre el front-end y back-end por medio de peticiones Ajax y el uso de API’s
- Aprendimos el uso y el gran aporte de la biblioteca multiplataforma JQuery, además de la forma de imprimir y hacer visual objetos en Canvas.
- Aprendimos a usar eventos en JavaScript.
- Además, también entendimos que puede haber una aplicación completamente aparte que funcione como componente visual para un sistema mientras se usa una API para asegurar su funcionamiento.
- Se puede dar una perspectiva al cliente de una manera eficiente, interactiva, dinámica, fácil de usar, en la que dicho usuario se siente satisfecho.