  
TECNOLÓGICO DE ESTUDIOS SUPERIORES DE ECATEPEC.  
  
INGENIERIA EN SISTEMAS COMPUTACIONALES.

**Integrantes:  
Actividades que realizaron**

|  |  |
| --- | --- |
| **Integrantes** | **Actividad** |
| **Alvarado Ayala Jorge Alexis** | * **Github** * **Propuesta de modulo pdf** |
| **Moreno Guzmán Angel David** | * **Propuesta de modulo pdf** * **Modulo pdf con imagen** |
| **Ruíz Romero Sandra Jacqueline** | * **Documentación** * **Búsqueda de propuestas en reportes pdf** |

**Equipo:** 404 Not Found

**Grupo:** 5801

**Materia:**

Bases de datos para dispositivos móviles.

**Profesor:**

Griselda Cortes Barrera.

**Avance proyecto**

Módulo de reportes.

**Módulo de reportes.**

* **Requerimientos:**
* Se requiere de un sistema genere un reporte de asistencia de la entrada y salida del personal e invitados a la empresa.
* Exportación de reportes mediante la librería jspdf, para convertir los reportes a un archivo pdf.
* El sistema tendrá un control de todos los reportes realizados, y se podrán ordenar y consultar.
* **Problemática**

La asistencia y puntualidad de los empleados o visitantes en una empresa son muy importantes porque se ven reflejadas en la productividad del personal dentro de la compañía, si un empleado está a tiempo en su lugar de trabajo podrá planear y ejecutar sus actividades diarias llevándolas a cabo de forma satisfactoria.  
  
En ocasiones resulta complicado mantener un control sobre la hora de entrada y salida del personal, y es aún más difícil de monitorear en empresas con un alto volumen de empleados, pero como en otros aspectos de la vida cotidiana, la tecnología ayuda a que estas actividades sean realizadas de forma sencilla, rápida y eficiente.

* **Objetivo general**

Hacer un control de asistencia (en tiempo real), nos permita capturar el momento preciso en que un trabajador hace su ingreso a su centro de labores, de igual forma cuando esta marca su salida, su característica principal es que lo hace en tiempo real lo que nos permitirá fácilmente detectar si este llegó tarde, no asistió o hizo horas extras. Y para los usuarios invitados se generará otro reporte exclusivo de visitantes.

* **Objetivos específicos.**

Para lograr el objetivo general, se tienen los siguientes objetivos específicos.

* Maquetado y diseño de la interfaz gráfica del módulo de reportes para trabajadores.
* Maquetado y diseño de la interfaz gráfica del módulo de reportes para usuarios invitados.
* Exportación de los reportes generados por el sistema, por medio de la librería jspdf.
* Acceso a la base de datos para la lectura de los datos.

**Propuesta**

**Librería 1 PDF Make.**

Esta librería escrita en Typescript es un wrapper basado en **pdf make** para generar documentos PDF de una manera fácil y legible.

## **Instalación**

Esta versión fue construida teniendo en cuenta **pdfmake@0.1.x.**

Para utilizar esta biblioteca es necesario instalar tanto **pdfmake-wrapper como** **pdfmake:**

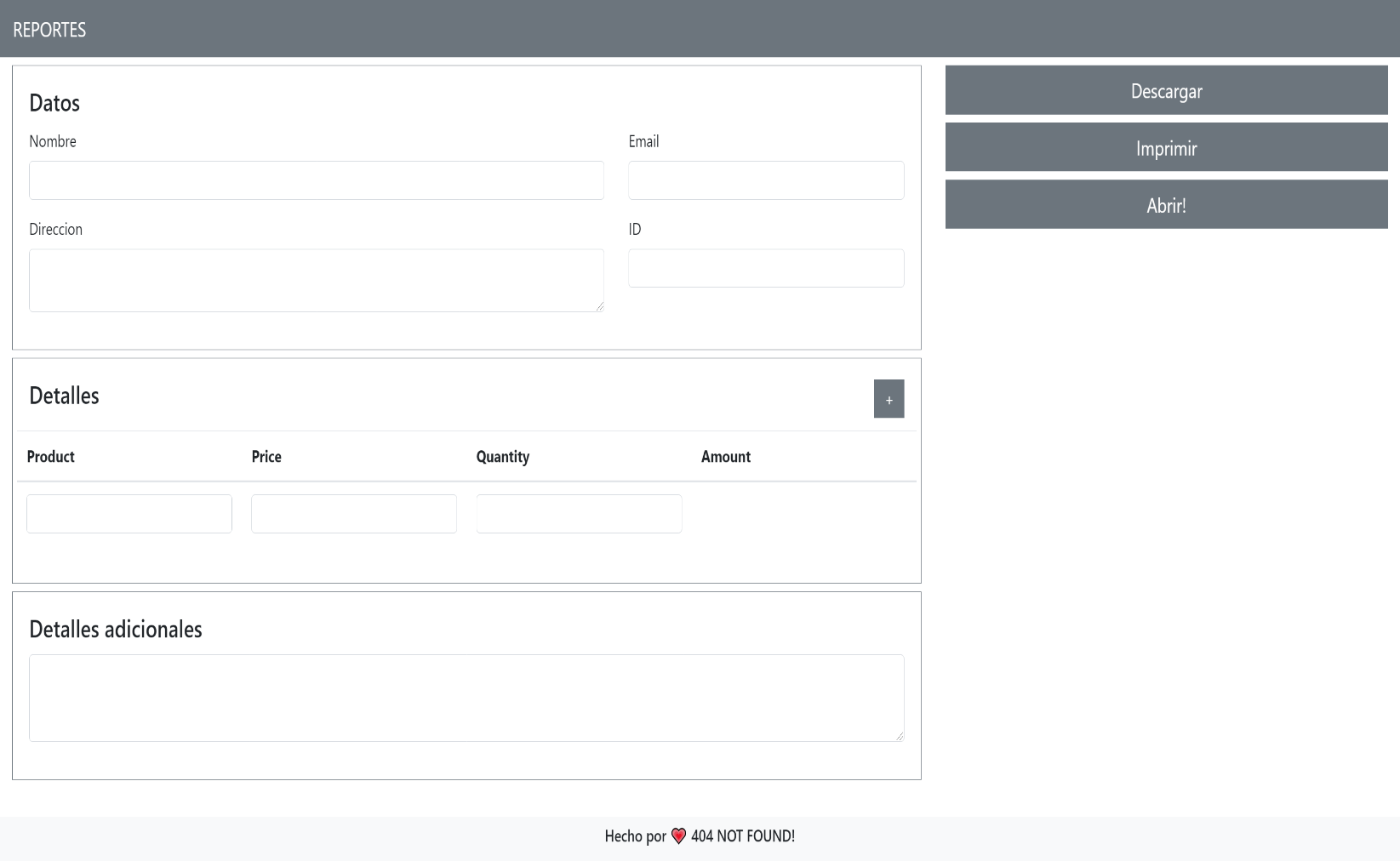
**$ npm instalar pdfmake --save**

y

**$ npm instalar pdfmake-wrapper –save**

**Diseño**

Este módulo, captura mediante el formulario los datos y mediante la librería **Pdf Make-Wrapper** nos permite la creación de documentos PDF, esto nos hará más fácil la creación de reportes e imprimirlos en ese formato.



**Código.**

Colocaremos una pequeña fracción de cada elemento de nuestro modulo, ya que es un poco extenso.

**App.component.html**

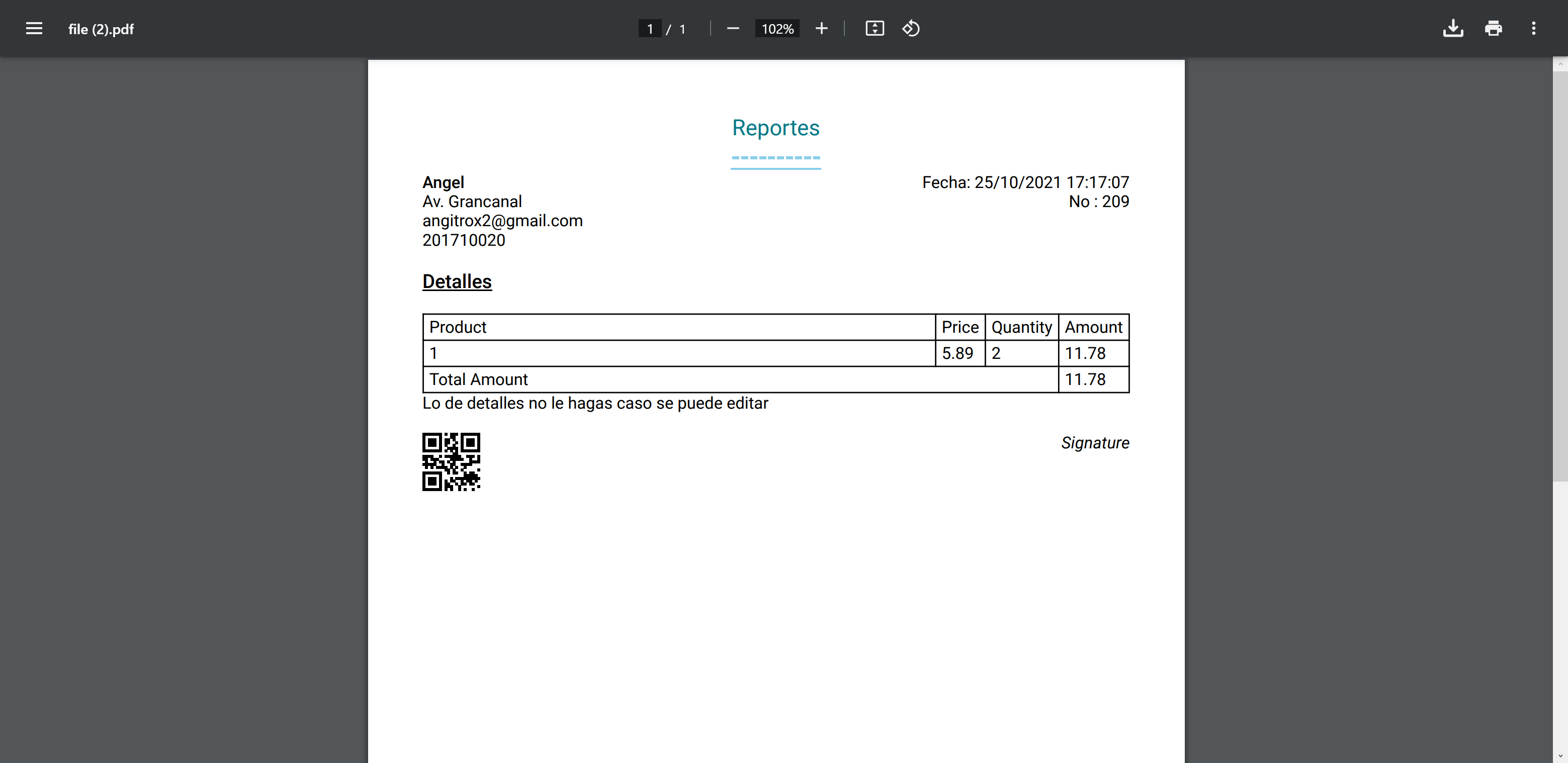
En app.component.html se coloca la vista de la aplicación “reportes”, la cual está conformada por un formulario el cual al llenarlo podemos generar el archivo PDF para poderlo descargarlo y a su vez imprimirlo.

nav class="navbar navbar-expand-sm navbar-dark bg-secondary d-flex justify-content-between">  
 <a class="navbar-brand" href="#">REPORTES</a>  
 </nav>  
<form #***invoiceForm***="ngForm">  
 <div class="container-fluid pt-2 mb-5">  
 <div class="row">  
 <div class="col-md-8">  
 <div class="card border-secondary">  
 <div class="card-body">  
 <h4 class="card-title">Datos</h4>  
 <div class="row">  
 <div class="col-md-8">  
 <div class="form-group">  
 <label for="name">Nombre</label>  
 <input type="text" class="form-control" name="name" id="name"  
 [ngClass]="{'is-invalid': ***invoiceForm***.submitted && ***CustName***.invalid}"  
 [(ngModel)]="invoice.customerName" #***CustName***="ngModel" required>  
 </div>  
 <div class="form-group">  
 <label for="address">Direccion</label>  
 <textarea class="form-control" name="address" id="address" row="3"  
 [ngClass]="{'is-invalid': ***invoiceForm***.submitted && ***Address***.invalid}"  
 [(ngModel)]="invoice.address" #***Address***="ngModel" required></textarea>  
 </div>  
 </div>  
 <div class="col-md-4">  
 <div class="form-group">  
 <label for="email">Email </label>  
 <input type="email" class="form-control" name="email" id="email"  
 [(ngModel)]="invoice.email" required>  
 </div>  
 <div class="form-group">  
 <label for="contactNo">ID</label>  
 <input type="number" class="form-control" name="contactNo" id="contactNo"  
 [ngClass]="{'is-invalid': ***invoiceForm***.submitted && ***ContactNo***.invalid}"  
 [(ngModel)]="invoice.contactNo" #***ContactNo***="ngModel" required>  
 </div>  
 </div>  
 </div>  
 </div>  
 </div>  
 <div class="card border-secondary mt-2">  
 <div class="card-body">  
 <h4 class="card-title d-flex justify-content-between">Detalles <button type="button"  
 class="btn btn-secondary" (click)="addProduct()">+</button></h4>  
 <div class="row">  
 <table class="table">  
 <thead>  
 <tr>  
 <th width="25%">Product</th>  
 <th width="25%">Price</th>  
 <th width="25%">Quantity</th>  
 <th width="25%">Amount</th>  
 </tr>  
 </thead>  
 <tbody>  
 <tr *\*ngFor*="let ***product*** of invoice.products; let ***i***=index">  
 <td scope="row">  
 <input type="text" class="form-control" name="productName{{***i***}}"  
 [ngClass]="{'is-invalid': ***invoiceForm***.submitted && ***ProductName***.invalid}"  
 id="productName" [(ngModel)]="***product***.name" #***ProductName***="ngModel"  
 required>  
 </td>  
 <td>  
 <input type="number" class="form-control" name="price{{***i***}}" id="price"  
 [ngClass]="{'is-invalid': ***invoiceForm***.submitted && ***ProductPrice***.invalid}"  
 [(ngModel)]="***product***.price" #***ProductPrice***="ngModel" required>  
 </td>  
 <td>  
 <input type="number" class="form-control" name="quantity{{***i***}}" id="quantity"  
 [ngClass]="{'is-invalid': ***invoiceForm***.submitted && ***ProductQty***.invalid}"  
 [(ngModel)]="***product***.qty" #***ProductQty***="ngModel" required>  
 </td>  
 <td>  
 {{***product***.price \* ***product***.qty || ''}}  
 </td>  
 </tr>  
 </tbody>  
 </table>  
 </div>  
 </div>  
 </div>  
 <div class="card border-secondary mt-2">  
 <div class="card-body">  
 <h4 class="card-title">Detalles adicionales</h4>  
 <div class="form-group">  
 <textarea class="form-control" name="additionalDetails" rows="3"  
 [(ngModel)]="invoice.additionalDetails"></textarea>  
 </div>  
 </div>  
 </div>  
  
 </div>  
 <div class="col-md-4">  
 <button type="button" class="btn btn-secondary btn-lg btn-block"  
 (click)="***invoiceForm***.valid ? generatePDF('download') : ''">Descargar</button>  
 <button type="button" class="btn btn-secondary btn-lg btn-block"  
 (click)="***invoiceForm***.valid ? generatePDF('print') : ''">Imprimir</button>  
 <button type="submit" class="btn btn-secondary btn-lg btn-block"  
 (click)="***invoiceForm***.valid ? generatePDF() : ''">Abrir!</button>  
 </div>  
 </div>  
 </div>  
</form>  
<footer class="fixed-bottom">  
 <div class="bg-light text-center p-2">  
 <h6>Hecho por 💗 404 NOT FOUND!  
  
 </h6>  
 </div>  
</footer>

* **App.component.ts**

En el componente .ts, se puede dar estructura al documento de PDF para darle una vista cuando se genere, al igual se declaran las clases que ocuparemos para guardar los diferentes campos de nuestro formulario.

import { ***Component*** } from '@angular/core';  
  
import pdfMake from "pdfmake/build/pdfmake";  
import pdfFonts from "pdfmake/build/vfs\_fonts";  
pdfMake.vfs = pdfFonts.pdfMake.vfs;  
  
class Product{  
 name: string;  
 price: number;  
 qty: number;  
}  
class Invoice{  
 customerName: string;  
 address: string;  
 contactNo: number;  
 email: string;  
  
 products: Product[] = [];  
 additionalDetails: string;  
  
 constructor(){  
 // Initially one empty product row we will show  
 this.products.push(new Product());  
 }  
}  
@Component({  
 selector: 'app-root',  
 templateUrl: './app.component.html',  
 styleUrls: ['./app.component.css']  
})  
export class AppComponent {  
 invoice = new Invoice();  
  
 generatePDF(action = 'open') {  
 let docDefinition = {  
 content: [  
 {  
 text: 'Reportes',  
 fontSize: 16,  
 alignment: 'center',  
 color: '#047886'  
 },  
 {  
 text: '----------',  
 fontSize: 20,  
 bold: true,  
 alignment: 'center',  
 decoration: 'underline',  
 color: 'skyblue'  
 },  
  
 {  
 columns: [  
 [  
 {  
 text: this.invoice.customerName,  
 bold:true  
 },  
 { text: this.invoice.address },  
 { text: this.invoice.email },  
 { text: this.invoice.contactNo }  
 ],  
 [  
 {  
 text: `Fecha: ${new ***Date***().toLocaleString()}`,  
 alignment: 'right'  
 },  
 {  
 text: ` No : ${((***Math***.random() \*1000).toFixed(0))}`,  
 alignment: 'right'  
 }  
 ]  
 ]  
 },  
 {  
 text: 'Detalles',  
 style: 'sectionHeader'  
 },  
 {  
 table: {  
 headerRows: 1,  
 widths: ['\*', 'auto', 'auto', 'auto'],  
 body: [  
 ['Product', 'Price', 'Quantity', 'Amount'],  
 ...this.invoice.products.map(p => ([p.name, p.price, p.qty, (p.price\*p.qty).toFixed(2)])),  
 [{text: 'Total Amount', colSpan: 3}, {}, {}, this.invoice.products.reduce((sum, p)=> sum + (p.qty \* p.price), 0).toFixed(2)]  
 ]  
 }  
 },  
  
 {  
 text: this.invoice.additionalDetails,  
 margin: [0, 0 ,0, 15]  
 },  
 {  
 //genera QR del nombre  
 columns: [  
 [{ qr: `${this.invoice.customerName}`, fit: '50' }],  
 [{ text: 'Signature', alignment: 'right', italics: true}],  
 ]  
 },  
  
 ],  
 styles: {  
 sectionHeader: {  
 bold: true,  
 decoration: 'underline',  
 fontSize: 14,  
 margin: [0, 15,0, 15]  
 }  
 }  
 };  
  
 if(action==='download'){  
 pdfMake.createPdf(docDefinition).download();  
 }else if(action === 'print'){  
 pdfMake.createPdf(docDefinition).print();  
 }else{  
 pdfMake.createPdf(docDefinition).open();  
 }  
  
 }  
  
 addProduct(){  
 this.invoice.products.push(new Product());  
 }  
  
}

**Vista de nuestro PDF generado.**

**Propuesta (esta será la que estaremos utilizando en el desarrollo de nuestro proyecto)**

**Librería 2: jsPDF.**

**Maquetado del diseño de los reportes a generar.**

Tabla

Descripción generada automáticamente Tabla

Descripción generada automáticamente

Aunque existan varias opciones, nosotros nos vamos a decantar por jsPDF que es un módulo basado en Javascript, principalmente utilizado para generar archivos PDF desde el lado cliente.

Nos ofrece una gran cantidad de métodos que nos va a permitir personalizar la vista del PDF a nuestras preferencias y gustos.

**1.- ¿Qué es un fichero PDF?**

PDF ([siglas](https://es.wikipedia.org/wiki/Sigla) en [inglés](https://es.wikipedia.org/wiki/Idioma_ingl%C3%A9s) de Portable Document Format, «formato de documento portátil») es un [formato de almacenamiento](https://es.wikipedia.org/wiki/Formato_de_almacenamiento) para documentos digitales independiente de plataformas de [software](https://es.wikipedia.org/wiki/Software) o [hardware](https://es.wikipedia.org/wiki/Hardware). Este formato es de tipo compuesto ([imagen vectorial](https://es.wikipedia.org/wiki/Imagen_vectorial), [mapa de bits](https://es.wikipedia.org/wiki/Mapa_de_bits) y [texto](https://es.wikipedia.org/wiki/Documento_de_texto)). (Definición sacada desde la Wikipedia)

Los ficheros pueden contener textos, imágenes, hipervínculos, video, botones interactivos y más.

**2.- Requisitos antes de empezar a trabajar con el proyecto:**

- NodeJS y NPM instalados en su última versión

- Angular CLI instalado.

**3.- Crear el proyecto y primer inicio**

- Creamos un nuevo proyecto.

**ng new angular-jspdf-master --routing**

**4.- Instalacion de la librería jsPDF dentro de nuestro proyecto.**

Con esta librería vamos a generar PDFs en nuestra vista Front-end, para instalarlo vamos a nuestra consola de comandos y escribimos el siguiente comando dentro de nuestro proyecto en angular a trabajar:

**npm install jspdf –save**

**HTML2Canvas**

Con la anterior librería generamos PDF, pero necesitamos que el PDF se genere desde el contenido que se encuentre en una capa o div HTML de nuestra vista Front, para ello usaremos esta librería que nos facilitará la tarea de renderizar el contenido HTML de nuestra vista, para instalarlo ejecutamos 2 comandos cada uno por separado

**// Primero ejecutamos este comando**

**npm install --save html2canvas**

**// Luego este otro**

**npm install --save @types/html2canvas**

npm install jspdf@1.5.3 html2canvas

npm install @types/jspdf @types/html2canvas -D // en dependencias desarrollo con "-D"

Una vez que hemos realizado las instalaciones, tenemos que añadir la referencias de los scripts de jsPDF y html2canvas, para poder utilizarlos en nuestro proyecto dentro del fichero angular.json:

Texto

Descripción generada automáticamente

Ahora que ya tenemos esto añadido, podremos usar los estilos de bootstrap y los scripts correspondientes para jsPDF y html2canvas, aunque en el caso de los scripts, para hacerlo sin problemas tenemos que configurar una propiedad dentro del tsconfig.json, para evitar un error. Seguimos en el siguiente punto para hacer este paso.

**5.- Configurar el apartado de importaciones desde módulos sin exportación**

Ahora para poder trabajar con jsPDF como html2canvas debemos de configurar dentro de la configuración de typescript de nuestro proyecto, el apartado ***allowSyntheticDefaultImports***.

Esto servirá para poder permitir importaciones predeterminadas desde módulos que no tienen exportación predeterminada. Esto no afectará a la emisión del código, simplemente para la verificación a la hora de compilar.

Vamos a ello y actualizamos ***tsconfig.base.json*** o ***tsconfig.json.***

Dependiendo de la versión, tendremos que usar una opción u otra. El primer fichero lo tenemos disponible con Angular 10 y el segundo con los proyectos de Angular 9 o menos.

Añadimos ***allowSyntheticDefaultImports***dentro ***compilerOptions***



## **Componente**

En nuestro componente llamado app.component.ts instanciamos las librerías que hemos instalado.



**5.- Creamos nuestro primer fichero PDF desde Angular.**

Teniendo todo instalado y configurado todo correctamente, cogemos el componente ***app.component.ts*** (podría ser otro que sea de nuestro agrado) y añadimos lo siguiente:

import { ***Component*** } from '@angular/core';  
import jsPDF from 'jspdf';  
import ***html2canvas*** from 'html2canvas';  
@Component({  
 selector: 'app-root',  
 templateUrl: './app.component.html',  
 styleUrls: ['./app.component.css'],  
})  
export class AppComponent {

/\*informacion fake para prueba\*/  
 DATOS\_EMPLEADOS = [  
 {  
 dni: 1,  
 name: 'Juan',  
 ap: 'Perez',  
 am: 'Rojas',  
 H\_E: '09:20 hrs',  
 H\_S: '17:00 hrs',  
 Date: '27/10/2021',  
 },  
 {  
 dni: 2,  
 name: 'Noemi',  
 ap: 'Perez',  
 am: 'Rojas',  
 H\_E: '09:20 hrs',  
 H\_S: '17:00 hrs',  
 Date: '27/10/2021',  
 },  
 {  
 dni: 3,  
 name: 'Denisse',  
 ap: 'Arteaga',  
 am: 'Romero',  
 H\_E: '09:20 hrs',  
 H\_S: '17:00 hrs',  
 Date: '28/10/2021',  
 },  
 {  
 dni: 4,  
 name: 'Carlos',  
 ap: 'Ortega',  
 am: 'Fuentes',  
 H\_E: '09:20 hrs',  
 H\_S: '17:00 hrs',  
 Date: '28/10/2021',  
 },  
 {  
 dni: 5,  
 name: 'Michel',  
 ap: 'Sosa',  
 am: 'Fuentes',  
 H\_E: '09:20 hrs',  
 H\_S: '17:00 hrs',  
 Date: '28/10/2021',  
 },  
 {  
 dni: 6,  
 name: 'Alan',  
 ap: 'Diaz',  
 am: 'Cortes',  
 H\_E: '09:00 hrs',  
 H\_S: '17:00 hrs',  
 Date: '28/10/2021',  
 },  
 ];  
 // tslint:disable-next-line:typedef  
 downloadPDF() {  
 const DATA = ***document***.getElementById('htmlData');  
 const doc = new jsPDF('p', 'pt', 'a4');  
 const options = {  
 background: 'white',  
 scale: 3  
 };  
 ***html2canvas***(DATA, options).then((canvas) => {  
  
 const img = canvas.toDataURL('image/PNG');  
  
 // Add image Canvas to PDF  
 const bufferX = 15;  
 const bufferY = 15;  
 const imgProps = (doc as any).getImageProperties(img);  
 const pdfWidth = doc.internal.pageSize.getWidth() - 2 \* bufferX;  
 const pdfHeight = (imgProps.height \* pdfWidth) / imgProps.width;  
 doc.addImage(img, 'PNG', bufferX, bufferY, pdfWidth, pdfHeight, undefined, 'FAST');  
 return doc;  
 }).then((docResult) => {  
 docResult.save(`${new ***Date***().toISOString()}\_tutorial.pdf`);  
 });  
 }  
}

Lo que hace ese componente es sencillo:

Tenemos un constructor que llama a una función “***downloadPdf()***” donde se crea en nuevo documento y mediante la función save, se habilita para descargar el fichero con el nombre “\_tutorial.pdf” .

Tenemos que tener en cuenta estos aspectos:

***const DATA = document.getElementById(‘htmlData’);***

Asignamos en “DATA” el contenido de la tabla y el título “Reporte asistencia…” para poder usarlo en la generación del PDF como la información que queremos mostrar.

***const doc = new jsPDF(‘p’, ‘pt’, ‘a4’);***

Iniciamos el valor “doc” que será el objeto del documento para generar el PDF añadiendo la orientación “p” (Portrait = vertical), la unidad de media como “pt” y el tamaño del PDF, “A4”.

***html2canvas(DATA, options).then((canvas) => {***

***const img = canvas.toDataURL(‘image/PNG’);***

**// Añadir imagen Canvas a PDF  
const bufferX = 15;  
const bufferY = 15;  
const imgProps = (doc as any).getImageProperties(img);  
const pdfWidth = doc.internal.pageSize.getWidth() — 2 \* bufferX;  
const pdfHeight = (imgProps.height \* pdfWidth) / imgProps.width;  
doc.addImage(img, ‘PNG’, bufferX, bufferY, pdfWidth, pdfHeight, undefined, ‘FAST’);  
return doc;  
}).then((docResult) => {  
docResult.save(‘Reporte.pdf’);  
});**

Este apartado es importante, ya que va a ser el punto donde vamos a crear una imagen mediante el canvas teniendo en cuenta la información que hemos obtenido desde el id “htmlData”.

Le vamos a asignar las propiedades de las página, ubicación y con ello en el resultado, creamos el fichero PDF con el nombre que especificamos.

**6.- Preparando el HTML que usaremos para el PDF**

Vamos a app.component.html y añadimos lo siguiente, para crear una tabla con la información de los usuarios, que después se obtendrá la información de una API o la BD del proyecto.

<div class="container">  
 <div class="row">  
 <div id="htmlData">  
 <h2>Reporte de asistencia</h2>  
 <table class="table table-bordered mt-4">  
 <tr class="table-primary">  
 <th>DNI</th>  
 <th>Name</th>  
 <th>A\_P</th>  
 <th>A\_M</th>  
 <th>Hora\_entrada</th>  
 <th>Hora\_salida</th>  
 <th>Fecha</th>  
  
 </tr>  
 <tr *\*ngFor*="let ***d\_e*** of DATOS\_EMPLEADOS">  
 <th>{{***d\_e***.dni}}</th>  
 <td>{{***d\_e***.name}}</td>  
 <td>{{***d\_e***.ap}}</td>  
 <td>{{***d\_e***.am}}</td>  
 <td>{{***d\_e***.H\_E}}</td>  
 <td>{{***d\_e***.H\_S}}</td>  
 <td>{{***d\_e***.Date}}</td>  
 </tr>  
 </table>  
 </div>  
 </div>  
</div>

Hay que prestar atención en el apartado donde añadimos el id=”***htmlData***” que será muy importante, ya que esté contenido será el que usemos para plasmar esa información dentro del PDF que vamos a crear.

Por ultimo corremos nuestra aplicación y este será el resultado.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Nos genera Nuestro reporte en formato PDF.

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja

**Reporte de asistencia con imagen.**

Aquí implementaremos una imagen ya que serán las imágenes que se recogerán de la bd para mostrarlas en nuestros reportes.

En el apartado de html vamos a colocar el siguiente código:

<div class="container" >  
 <div class="row" >  
 <div id="htmlData">  
 <h2>Reporte de asistencia</h2>  
 <table class="table table-striped table-hover">  
 <thead class="table-dark">  
 <tr border border-white>  
 <th>DNI</th>  
 <th>Imagen</th>  
 <th>Name</th>  
 <th>A\_P</th>  
 <th>A\_M</th>  
 <th>Hora\_entrada</th>  
 <th>Hora\_salida</th>  
 <th>Fecha</th>  
 </tr>  
 </thead>  
 <tr *\*ngFor*="let ***d\_e*** of DATOS\_EMPLEADOS">  
 <td>{{***d\_e***.dni}}</td>  
 <td><img src="{{***d\_e***.img}}" width="100" height="100"></td>  
 <td>{{***d\_e***.name}}</td>  
 <td>{{***d\_e***.ap}}</td>  
 <td>{{***d\_e***.am}}</td>  
 <td>{{***d\_e***.H\_E}}</td>  
 <td>{{***d\_e***.H\_S}}</td>  
 <td>{{***d\_e***.Date}}</td>  
 </tr>  
 </table>  
 </div>  
 </div>  
</div>  
  
  
<div align="center">  
 <button class="btn col col-lg-9 btn-danger btn-block ml-4" (click)="downloadPDF()">Download PDF</button>  
</div>

Ahora, en la parte de **app.component.ts** vamos a colocar el siguiente código, que será el que nos permitirá insertar las imágenes a nuestro reporte en formato pdf.

import { ***Component*** } from '@angular/core';  
import jsPDF from 'jspdf';  
import ***html2canvas*** from 'html2canvas';  
@Component({  
 selector: 'app-root',  
 templateUrl: './app.component.html',  
 styleUrls: ['./app.component.css'],  
})  
export class AppComponent {  
 DATOS\_EMPLEADOS = [  
 {  
 dni: 1,  
 img: 'assets/user.png',  
 ap: 'Perez',  
 am: 'Rojas',  
 H\_E: '09:20 hrs',  
 H\_S: '17:00 hrs',  
 Date: '27/10/2021',  
 },  
 {  
 dni: 2,  
 img: 'assets/user1.png',  
 name: 'Noemi',  
 ap: 'Perez',  
 am: 'Rojas',  
 H\_E: '09:20 hrs',  
 H\_S: '17:00 hrs',  
 Date: '27/10/2021',  
 },  
 {  
 dni: 3,  
 img: 'assets/user1.png',  
 name: 'Denisse',  
 ap: 'Arteaga',  
 am: 'Romero',  
 H\_E: '09:20 hrs',  
 H\_S: '17:00 hrs',  
 Date: '28/10/2021',  
 },  
 {  
 dni: 4,  
 img: 'assets/user.png',  
 name: 'Carlos',  
 ap: 'Ortega',  
 am: 'Fuentes',  
 H\_E: '09:20 hrs',  
 H\_S: '17:00 hrs',  
 Date: '28/10/2021',  
 },  
 {  
 dni: 5,  
 img: 'assets/user.png',  
 name: 'Michel',  
 ap: 'Sosa',  
 am: 'Fuentes',  
 H\_E: '09:20 hrs',  
 H\_S: '17:00 hrs',  
 Date: '28/10/2021',  
 },  
 {  
 dni: 6,  
 img: 'assets/user.png',  
 name: 'Alan',  
 ap: 'Diaz',  
 am: 'Cortes',  
 H\_E: '09:00 hrs',  
 H\_S: '17:00 hrs',  
 Date: '28/10/2021',  
 },  
 ];  
 // tslint:disable-next-line:typedef  
 downloadPDF() {  
 const DATA = ***document***.getElementById('htmlData');  
 const doc = new jsPDF('p', 'pt', 'a4');  
 const options = {  
 background: 'white',  
 scale: 3  
 };  
 ***html2canvas***(DATA, options).then((canvas) => {  
  
 const img = canvas.toDataURL('image/PNG');  
  
 // Add image Canvas to PDF  
 const bufferX = 15;  
 const bufferY = 15;  
 const imgProps = (doc as any).getImageProperties(img);  
 const pdfWidth = doc.internal.pageSize.getWidth() - 2 \* bufferX;  
 const pdfHeight = (imgProps.height \* pdfWidth) / imgProps.width;  
 doc.addImage(img, 'PNG', bufferX, bufferY, pdfWidth, pdfHeight, undefined, 'FAST');  
 return doc;  
 }).then((docResult) => {  
 docResult.save(`${new ***Date***().toISOString()}\_tutorial.pdf`);  
 });  
 }  
}

**Vista del modulo reporte con imagen**



GITHUB

Subimos nuestro modulo de reportes en una nueva rama llamada reportes en el repositorio general creado por nuestra compañera Sandra.

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamente

Texto

Descripción generada automáticamenteTexto

Descripción generada automáticamenteTexto

Descripción generada automáticamenteUna captura de pantalla de una computadora

Descripción generada automáticamente

Como se ve en la imagen el repositorio ha quedado agregado, realizando los cambios de la implementación de la imagen