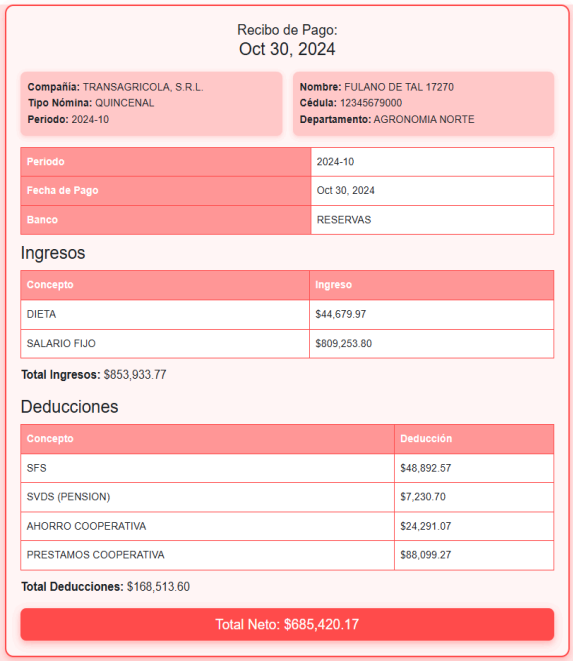
Aplicación web de Registro de Recibos

Esta página se trata de un registro de los recibos de pago de nómina de los empleados, dentro de la cuál el usuario puede ingresar, verificar e imprimir los últimos pagos realizados a su nombre.



La pantalla principal compone el formato de los recibos, con un diseño moderno y apacible a la vista, y que directamente mostrará el último recibo de pago del empleado ingresado. Además de esto, en la esquina superior derecha tendrá un menú con las fechas de pago anteriores, las cuales si son seleccionadas reemplazarán la información dentro del recibo por la información del recibo con la fecha seleccionada.

La página, como forma de reconocer el usuario, obtiene el nombre de usuario de Windows del dispositivo del usuario, y se encarga de relacionarlo con el código del empleado almacenado en la base de datos, de la cual se obtiene la información deseada.

Programación de la aplicación

Tecnologías utilizadas:

Frontend

- Angular Framework (version 19.0.1)
- Node.js (version 23.9.0)
- Editor de código: Visual Studio Code

Asegurarse de que el proyecto en Angular tenga los siguientes paquetes:

```
├── @angular-devkit/build-angular@19.0.2
├── @angular/animations@19.0.1
├── @angular/cdk@19.0.1
├── @angular/cli@19.0.2
├── @angular/common@19.0.1
├── @angular/compiler-cli@19.0.1
├── @angular/compiler@19.0.1
├── @angular/core@19.0.1
├── @angular/forms@19.0.1
├── @angular/material@19.0.1
├── @angular/platform-browser-dynamic@19.0.1
├── @angular/platform-browser@19.0.1
├── @angular/router@19.0.1
├── @electron/remote@2.1.2
├── @types/jasmine@5.1.5
├── boostrap@2.0.0
├── electron@34.1.0
├── ini@5.0.0
├── jasmine-core@5.4.0
├── karma-chrome-launcher@3.2.0
├── karma-coverage@2.2.1
├── karma-jasmine-html-reporter@2.1.0
├── karma-jasmine@5.1.0
├── karma@6.4.4
├── rxjs@7.8.1
├── tslib@2.8.1
├── typescript@5.6.3
└── zone.js@0.15.0
```

Backend

- ASP.Net (net 9.0)
- Herramienta: Visual Studio

Asegurarse que la API tenga los siguientes paquetes:

> Microsoft.AspNetCore.Authentication.Negotiate
> Microsoft.AspNetCore.OpenApi
> Microsoft.EntityFrameworkCore.Design
> Microsoft.EntityFrameworkCore.SqlServer
> Microsoft.EntityFrameworkCore.Tools
> Microsoft.Extensions.Configuration.Ini
> Swashbuckle.AspNetCore
> System.Configuration.ConfigurationManager

Tome en cuenta que es posible crear, manejar y ejecutar ambas partes de la aplicación de manera efectiva solamente en Visual Studio Code.

Descripción de Procesos

Ambas partes se ejecutan de manera separada, pero trabajan al unísono. La API se encarga de realizar las consultas y filtrar la información desde la base de datos, y mantenerla disponible para que Angular pueda acceder a la misma. Por otro lado, Angular se encarga de obtener la información desde la API, realizar un filtrado más profundo, y distribuir y mostrar la información deseada.

Procesos de la API:

Controlador (RecibosController.cs)

1. **Obtener el Usuario de Windows**

```
[HttpGet("access")]
    public IActionResult CheckUserAccess()
    {
      try
  {
    // Obtiene la identidad el usuario autenticada del HTTP context
    var identity = HttpContext.User.Identity;

    if (identity == null || !identity.IsAuthenticated)
    {
      return Unauthorized("No se encontró usuario autenticado");
    }

    return Ok(new
    {
      username = identity.Name,
      authenticationType = identity.AuthenticationType,
      isAuthenticated = identity.IsAuthenticated
    });
  }
  catch (Exception ex)
  {
    return BadRequest($"Error con el usuario: {ex.Message}");
    }
```

Se mantiene en espera, y es ejecutada por Angular. Retorna el usuario de Windows del dispositivo que ingresa desde Angular

2. **Obtener Último Recibo**

```
[HttpGet]
        public async Task<ActionResult<List<Recibos>>> RecibosEmpleado([FromQuery]
string codEmpleado)
        {

            Console.WriteLine($"Valor recibido en codEmpleado: {codEmpleado}");
            try
            {
                var ultRecibo = await context.Recibos
                                        .FromSqlInterpolated($"EXEC UltimoReciboEmpleado
@cod_empleado = {codEmpleado}")
                                        .ToListAsync();

                if (ultRecibo == null || ultRecibo.Count == 0)
                {
                    return NotFound(new { Message = $"No se encontraron recibos para el
empleado con código {codEmpleado}" });
                }

                return ultRecibo;
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Error[RecibosEmpleado]: {ex}");
                return StatusCode(500, new { Message = "Error al ejecutar el procedimiento
almacenado UltimoReciboEmpleado", Error = ex.Message });
            }
        }
```

Esta consulta se realiza automáticamente al iniciar la aplicación y se queda en espera. Angular, inmediatamente al obtener el usuario de Windows, le otorga el usuario a la consulta (string codEmpleado), y desde ahí se ejecuta "UltimoReciboEmpleado", la cual se explica más adelante, pero que básicamente, según el usuario, retorna su información.

3. **Obtener y Mostrar los Recibos Más Recientes**

```
[HttpGet("ultimos")]
        public async Task<ActionResult<List<RecibosRecientesDTO>>>
ReciboRecientesEmpleado([FromQuery] string codEmpleado)
        {
           try
           {
              var recibosRecientes = await context.RecibosRecientesDTO
                                    .FromSqlInterpolated($"EXEC ReciboRecientesEmpleado
@cod_empleado = {codEmpleado}")
                                    .ToListAsync();
              if (recibosRecientes == null || recibosRecientes.Count == 0)
              {
                 return NotFound(new { Message = $"No se encontraron recibos para el
empleado {codEmpleado}" });
              }
              return recibosRecientes;
           }
           catch (Exception ex)
           {
              Console.WriteLine($"Error[RecibosRecientesEmpleado]: {ex}");
              return StatusCode(500, new { Message = "Error al ejecutar el procedimiento
almacenado ReciboRecientesEmpleado ", Error = ex.Message });
           }
        }

[HttpGet("mostrar-ultimos")]
      public async Task<ActionResult<List<Recibos>>>
MostrarReciboRecienteEmpleado([FromQuery] string codEmpleado, DateOnly fechaPago)
      {
         try
         {
            var recibos = await context.Recibos
                                 .FromSqlInterpolated($"EXEC MostrarReciboRecienteEmpleado
@cod_empleado = {codEmpleado}, @fechaPago = {fechaPago}")
                                 .ToListAsync();
            if (recibos == null || recibos.Count == 0)
            {
               return NotFound(new { Message = $"No se encontraron recibos para el empleado
{codEmpleado} con fecha {fechaPago}" });
```

```
            }
            return recibos;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error[MostrarReciboRecientesEmpleado]: {ex}");
            return StatusCode(500, new { Message = "Error al ejecutar el procedimiento
almacenado MostrarReciboRecienteEmpleado", Error = ex.Message });
        }
    }
```

Luego de que Angular obtiene el código del usuario correctamente y despliega el último recibo, procede a llamar y realizar las consultas de los recibos más recientes. La consulta "ultimos" obtiene toda la información de los últimos 6 recibos registrados al nombre del usuario registrado, de los cuales solo devuelve la fecha y el código de período de nómina. Entonces, la fecha es desplegada en el frontend. Desde ahí, la fecha se almacena como una variable en el mismo frontend, y se le es enviada a la siguiente consulta, "mostrar-ultimos", que se encarga de obtener y mostrarla información del recibo según la fecha proporcionada.

Program.cs

```
using api.Services;
using Microsoft.EntityFrameworkCore;
using Microsoft.AspNetCore.Authentication.Negotiate;
using System.Configuration;
using Microsoft.AspNetCore.Hosting;


var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

var allowedOrigins = builder.Configuration.GetValue<string>("allowedOrigins")!.Split(",");

builder.Configuration.AddJsonFile("appsettings.json", optional: false, reloadOnChange: true);
```

```
// Obtener el ConnectionString desde la configuración
var connectionString = builder.Configuration.GetConnectionString("RecibosApi");


builder.Services.AddDbContext<ApplicationDBContext>(options =>
{
   options.UseSqlServer(connectionString);
});

builder.Services.AddControllers().AddJsonOptions(options =>
{
   options.JsonSerializerOptions.PropertyNamingPolicy = null;
});

// Configurar CORS
builder.Services.AddCors(options =>
{
   options.AddPolicy("AllowAngularApp",
       policy => policy.WithOrigins("http://10.20.30.16:4200", "http://localhost:4200") // Permitir
Angular
                   .AllowAnyMethod() // Permitir todos los métodos (GET, POST, etc.)
                   .AllowAnyHeader() // Permitir todos los encabezados
                   .AllowCredentials()
   );
});


builder.Services.AddAuthentication(NegotiateDefaults.AuthenticationScheme)
   .AddNegotiate();

builder.Services.AddAuthorization(options =>
{
   options.FallbackPolicy = options.DefaultPolicy;
});

var app = builder.Build();
```

```
app.UseCors("AllowAngularApp");
app.UseAuthentication();
app.UseAuthorization();

app.MapGet("/prueba", () => "Servidor en funcionamiento");

app.MapControllers();




app.Run();

public partial class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureAppConfiguration((context, config) =>
            {
                // Add the config.ini file to the configuration
                config.SetBasePath(Directory.GetCurrentDirectory());
                config.AddIniFile("config.ini", optional: true, reloadOnChange: true);
            })
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.ConfigureKestrel((context, options) =>
                {
                    // Read the BaseUrl from the configuration
                    var baseUrl = context.Configuration["Server:BaseUrl"];
                    if (!string.IsNullOrEmpty(baseUrl))
                    {
                        options.ListenLocalhost(new Uri(baseUrl).Port);
                    }
                });
```

```
            webBuilder.UseStartup<IStartup>();
        });
}
```

Este archivo contiene la configuración general de la API, como la autenticación, la autorización, el sistema de CORS y las posibles rutas permitidas de Angular, y demás.

api.csproj

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net9.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
                <PackageReference Include="Microsoft.AspNetCore.Authentication.Negotiate"
Version="9.0.2" />
    <PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="9.0.2" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="9.0.2">
                        <IncludeAssets>runtime; build; native; contentfiles; analyzers;
buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="9.0.2" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="9.0.2">
                        <IncludeAssets>runtime; build; native; contentfiles; analyzers;
buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.Extensions.Configuration.Ini" Version="9.0.3" />
    <PackageReference Include="Swashbuckle.AspNetCore" Version="6.6.2" />
     <PackageReference Include="System.Configuration.ConfigurationManager" Version="9.0.2"
/>
    <None Update="config.ini">
       <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
```

```
    </None>

  </ItemGroup>

</Project>
```

En este archivo se ponen a funcionar los paquetes instalados. Asegurarse de que los paquetes mencionados más arriba se encuentren referenciados en este archivo.

appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
                                    "ConnectionStrings":                    {                    "RecibosApi":
"Server='M3lv1n\\SQLEXPRESS01';Database=Recibos_Pago;Trusted_Connection=True;TrustS
erverCertificate=True;Integrated Security=True" },
  "allowedOrigins": "http://localhost:4200",
  "Authentication": {
    "Schemes": {
      "Windows": {
        "Enabled": true
      },
      "Negotiate" : {
        "Enabled": false
      }
    }
  }
}
```

En este archivo se define y establece la conexión de la API con el servidor y la base de datos. Esto incluye la especificación que le damos a la api para que acepte autenticación de Windows y Negotiate

Procesos de Angular

Servicio “Recibos.Service”

```
import { HttpClient, HttpHeaders, HttpParams } from '@angular/common/http';
import { inject, Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { Recibo } from './recibos.models';
import { environment } from '../environments/environment';
interface UsernameResponse {
  windowsIdentityName: string;
}

interface AccessResponse {
  username: string;
  isAuthorized: boolean;
}
@Injectable({
  providedIn: 'root',
})
export class RecibosService {
  //private apiURL = `http://localhost:7211/api/Recibos`;


  constructor(private http: HttpClient) {
  }

  getRecibos(codEmpleado: string): Observable<Recibo[]> {
    const params = new HttpParams().set('codEmpleado', codEmpleado.toString());
    console.log('hola, URL generada con parámetros:',
`${environment.apiURL}?${params.toString()}`);
    console.log("hola");
    return this.http.get<Recibo[]>(environment.apiURL, { params, withCredentials: true });
  }

  getReciboRecientePorFecha(codEmpleado: string, fechaPago: string): Observable<Recibo[]> {
    const url = `${environment.apiURL}/mostrar-ultimos`;
    const params = new HttpParams()
      .set('codEmpleado', codEmpleado.toString())
      .set('fechaPago', fechaPago);
```

```
    console.log(`URL generada: ${url}?${params.toString()}`);

    return this.http.get<Recibo[]>(url, { params, withCredentials: true });
  }

  getRecibosRecientes(codEmpleado: string): Observable<Recibo[]> {
    const url = `${environment.apiURL}/ultimos`;
    const params = new HttpParams().set('codEmpleado', codEmpleado.toString());
    console.log(`URL generada: ${url}?${params.toString()}`);

    return this.http.get<Recibo[]>(url, { params, withCredentials: true });
  }
  getWindowsUsername(): Observable<{username: string}> {
    return this.http.get<{username: string}>(`${environment.apiURL}/access`, {withCredentials:
true});
  }
}
```

Este es el servicio principal de la aplicación. Aquí se realizan todas las consultas requeridas a la API, almacenadas como funciones. Luego, las funciones se llaman y almacenan en variables donde se necesiten.

- getWindowsUsername() obtiene el nombre de usuario de windows el empleado ingresado
- La función getRecibos(), luego de reconocer el usuario, obtiene el último recibo del empleado, para así filtrar y distribuirlo en la página.
- getRecibosRecientes() obtiene LAS FECHAS de los últimos recibos del empleado ingresado, para así desplegarlos en el menú y hacer posible su elección.
- A la función getRecibosRecientesPorFecha(), se le envía la fecha seleccionada en el menú, y devuelve el recibo correspondiente a esa fecha.

Componente “Login”

Login.component.html

```
<div class="login-container">
  <div class="recibo">
    <div class="header">
      <h1>Bienvenido</h1>

    </div>
```

```
      <div class="login-form">
        <label for="codigoEmpleado"></label>
        <button (click)="iniciarSesion()">Acceder a Ultimo Recibo</button>
      </div>
    </div>
  </div>
```

login.component.ts

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { AuthService } from '../auth.service';
import { FormsModule } from '@angular/forms';
import { RecibosService } from '../recibos.service';
import { ConfigService } from '../config.service';
import { environment } from '../../environments/environment';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css'],
  imports: [FormsModule],
})
export class LoginComponent {
  codEmpleadoInput: number = 0;
  response: string = '';
  usuario: string | null = null;



constructor(
    private router: Router,
    private authService: AuthService,
    private recibosService: RecibosService,
    private configService: ConfigService
  ) {}

  ngOnInit(): void {
    const config = this.configService.getConfig();
    console.log(
      'Base URL:',
      config ? config['BaseUrl'] : 'Config not loaded from login'
```

```
    );
    console.log('Environment API URL:', environment.apiURL);
    this.recibosService.getWindowsUsername().subscribe(
      (response) => {
        this.usuario = response.username;
        console.log(this.usuario);
        console.log(typeof response);
        this.iniciarSesion();
      },
      (error) => {
        console.error('Error fetching username:', error);
      }
    );
  }
  iniciarSesion() {
    if (!this.usuario) {
      alert('Por favor, ingrese un código de empleado válido.');
      return;
    }

    this.authService.setUsuario(this.usuario);
    this.router.navigate(['/landing']); // Redirigir a la página de recibo
  }
}
```

En el componente de login, se realiza la verificación y obtención del usuario que ingresa a la página. En sí, el componente es dedicado específicamente para tratar al usuario, y no para mostrar nada en específico, debido a que está configurada para dirigirse al componente principal una vez obtenga el usuario. Debido a esto, si esto no procede, se mostrará la página del login, y se detallará un error en la consola.

Componente “Landing”

landing.component.ts

```
import {
  Component,
  ElementRef,
  inject,
```

```
  OnInit,
  ViewChild,
} from '@angular/core';
import { RecibosService } from '../recibos.service';
import { CommonModule } from '@angular/common'; // Asegúrate de esto
import { FormsModule } from '@angular/forms';
import { AuthService } from '../auth.service';
import { Router, RouterModule } from '@angular/router';
import { ConfigService } from '../config.service';

@Component({
  selector: 'app-landing',
  imports: [CommonModule, FormsModule, RouterModule],
  templateUrl: './landing.component.html',
  styleUrl: './landing.component.css',
})
export class LandingComponent implements OnInit {
  recibos: any[] = [];
  reciboActual: any = {};
  recibosRecientes: any[] = [];
  recibosRecientesPorFecha: any[] = [];
  tablaIngresos: any[] = [];
  tablaDeducciones: any[] = [];
  reciboEspecifico: any | null = null;
  totalIngresos: number = 0;
  totalDeducciones: number = 0;
  totalNeto: number = 0;

  fechaPago: string | null = null;

  codEmpleadoInput: string | null = null; // Campo de entrada del usuario

  isLoading: boolean = true;
  loading: boolean = true;
  username: string = '';
  isAuthorized: boolean = false;
  deviceInfo: any = {};
  user: string | null = null;
  usuario: string | null = null;
```

```
  constructor(
    private reciboService: RecibosService,
    private authService: AuthService,
    private router: Router,
    private configService: ConfigService
  ) {}
  @ViewChild('contenido', { static: false }) contenidoComponent!: ElementRef; // Accede al componente hijo

  imprimir() {
    const contenidoImprimir = this.contenidoComponent.nativeElement; // Accede al contenido del componente hijo
    const ventanaImpresion = window.open('', '', 'width=800, height=600');

    ventanaImpresion?.document.write(
      '<html><head><title>Imprimir</title></head><body>'
    );
    ventanaImpresion?.document.write(contenidoImprimir.innerHTML); // Inserta el contenido del componente hijo
    ventanaImpresion?.document.write('</body></html>');
    ventanaImpresion?.document.close();
    ventanaImpresion?.print();
  }
  ngOnInit(): void {
    this.codEmpleadoInput = this.authService.getUsuario(); // Recuperar el código desde el servicio
    this.fechaPago = this.authService.getFechaPago();
    const confi = this.configService.loadConfig();
    console.log('el confi', confi);
    const config = this.configService.getConfig();
    console.log(
      'Base URL:',
      config ? config['BaseUrl'] : 'Config not loaded in landing.component'
    );

    if (this.codEmpleadoInput) {
      this.buscarRecibos();
    } else {
      alert(
        'Error: No se encontró el código de empleado. Vuelva a iniciar sesión.'
```

```
    );
  }
}

buscarRecibos(): void {
  if (!this.codEmpleadoInput) {
    alert('Por favor, ingrese un código de empleado válido.');
    return;
  }

  // Get Windows username from backend
  this.reciboService.getWindowsUsername().subscribe(
    (response) => {
      this.codEmpleadoInput = response.username;
      console.log(`${response}`);
    },
    (error) => {
      console.error('Error fetching username:', error);
    }
  );

  this.reciboService.getRecibos(this.codEmpleadoInput).subscribe(
    (data) => {
      this.recibos = data;
      console.log('Recibos:', this.recibos);
      this.isLoading = false;
      this.procesarDatos();
      this.calcularTotales();
    },
    (error) => {
      console.error('Error al obtener los recibos:', error);
      this.isLoading = false;
    }
  );
  this.reciboService.getRecibosRecientes(this.codEmpleadoInput).subscribe(
    (data) => {
      this.recibosRecientes = data;
      console.log('Recibos Recientes:', this.recibosRecientes);
    },
    (error) => {
```

```
      console.log('Error con los recibos recientes', error);
    }
  );
}

seleccionarFecha(fecha: string) {
  this.fechaPago = fecha; // Asigna el valor seleccionado a fechaPago
  console.log('Fecha seleccionada:', this.fechaPago);
  this.buscarRecibosRecientesPorFecha();
}

buscarRecibosRecientesPorFecha(): void {
  if (!this.codEmpleadoInput) {
    alert('Por favor, ingrese un código de empleado válido.');
    return;
  }
  if (!this.fechaPago) {
    alert('falta fecha');
    return;
  }
  this.reciboService
    .getReciboRecientePorFecha(this.codEmpleadoInput, this.fechaPago)
    .subscribe(
      (data) => {
        this.recibos = data;
        console.log('Recibos Recientes Por Fecha:', this.recibos);
        this.procesarDatos();
        this.calcularTotales();
      },
      (error) => {
        console.log('Error con los recibos por fecha', error);
      }
    );
}

procesarDatos() {
  const mapaIngresos: { [key: string]: any } = {};
  const mapaDeducciones: { [key: string]: any } = {};

  this.recibos.forEach((recibo) => {
```

```
    const concepto = recibo.CONCEPTONOMINA;

    if (recibo.INGRESO > 0) {
      if (!mapaIngresos[concepto]) {
        mapaIngresos[concepto] = { concepto, ingreso: 0, deduccion: 0 };
      }
      mapaIngresos[concepto].ingreso += recibo.INGRESO;
    }

    if (recibo.DEDUCCION > 0) {
      if (!mapaDeducciones[concepto]) {
        mapaDeducciones[concepto] = { concepto, ingreso: 0, deduccion: 0 };
      }
      mapaDeducciones[concepto].deduccion += recibo.DEDUCCION;
    }
  });

  this.tablaIngresos = Object.values(mapaIngresos);
  this.tablaDeducciones = Object.values(mapaDeducciones);
}

calcularTotales() {
  this.totalIngresos = this.tablaIngresos.reduce(
    (total, item) => total + item.ingreso,
    0
  );
  this.totalDeducciones = this.tablaDeducciones.reduce(
    (total, item) => total + item.deduccion,
    0
  );
  this.totalNeto = this.totalIngresos - this.totalDeducciones;
}
menuAbierto: boolean = false;

toggleMenu() {
  this.menuAbierto = !this.menuAbierto;
}
getUserInfo() {
  return {
    userAgent: navigator.userAgent,
```

```
      platform: navigator.platform,
      language: navigator.language,
    };
  }
}
```

Esta es la parte del proyecto donde ocurre todo lo esencial que ocurre en la visualización del recibo. Se maneja toda la información de la api, obtenida por recibos.service, y se almacena de manera que pueda ser accedida en el html.

- ngOnInit(): Una vez la aplicación es iniciada, se toma la información necesaria del empleado desde el backend, y una vez la obtiene, inicia la función principal.
- buscarRecibos(): La función principal; ejecuta las funciones de obtener el último recibo y la lista de los últimos recibos según las fechas. Es totalmente dependiente de si ngOnInit() obtiene el código del empleado.
- seleccionarFecha() y buscarRecibosRecientesPorFecha(): Son excepciones de la automatización, ya que son iniciados en el momento que se abre el menú. SeleccionarFecha() se encarga únicamente de enviar la fecha seleccionada a la función correspondiente, mientras que BuscarRecibosRecientesPorFecha se encarga de hacer dos funciones: Mostrar las últimas fechas de pago y, luego de recibir la fecha seleccionada, intercambiar la información del recibo desplegado por la información del recibo según la fecha seleccionada.
- procesarDatos(): y calcularTotales()Es la parte del código que maneja los ingresos y las deducciones del recibo en cuestión, ya sea el último o el seleccionado según la fecha, distribuyendo la información en la plantilla del recibo, y calculando el total neto.