# Backend ucugestion - Tareas #17419
# Ajustar responsabilidades de servicio de pdf

15/10/2025 20:50 - Nicolas Arquiel

| | | | |
|---|---|---|---|
| **Estado:** | Nueva | **Fecha de inicio:** | 15/10/2025 |
| **Prioridad:** | Normal | **Fecha fin:** | |
| **Asignado a:** | | **% Realizado:** | 0% |
| **Categoría:** | | **Tiempo estimado:** | 0.00 hora |
| **Versión prevista:** | | **Tiempo dedicado:** | 0.00 hora |

**Descripción**

Ajustar responsabilidades de servicio de pdf

la funcion de generateSolicitudInscripcion tiene mucha logica que la deberia tener el modulo de alumnos

## Histórico

**#1 - 15/10/2025 20:50 - Nicolas Arquiel**

```
async generateSolicitudInscripcion(
  data: any,
  registrarEnBD = false,
  datosCertificado?: any,
  created_by?: string
): Promise<{ buffer: Buffer; filename: string }> {
  // Ya no necesitamos hacer transformaciones aquí
  // Los datos vienen completamente transformados

  return this.generateCertificadoSegunTipo(
    'solicitud-inscripcion',
    data,
    8,
    registrarEnBD,
    datosCertificado || data,
    created_by
  );
}
```

**#2 - 15/10/2025 20:51 - Nicolas Arquiel**

```
async generarPlanillaSolicidInscripcion(idalumno) {
  const queryRunner = this.dataSource.createQueryRunner();
  await queryRunner.connect();
  await queryRunner.startTransaction();

  try {
    // 1. DATOS DE LA CARRERA
    const [datosCarrera] = await queryRunner.query(`
      SELECT
        per.idpersona,
        car.nombre as carrera,
```

```
        car.idcarrera,
        pe.nombre as plan,
        s.nombre as sede,
        IF(a.idsede = 18, 'Virtual', 'Presencial') as modalidad,
        cl.ciclo as ciclo_lectivo,
        DATE_FORMAT(a.fechaingreso, '%d/%m/%Y') as fecha_inscripcion,
        a.idsedeinscripcion
      FROM alumnos a
      JOIN personas per ON a.idpersona = per.idpersona
      JOIN planes_estudios pe ON pe.idplanestudio = a.idplanestudio
      JOIN carreras car ON car.idcarrera = pe.idcarrera
      JOIN sedes s ON s.idsede = a.idsedeinscripcion
      JOIN ciclos_lectivos cl ON cl.id = a.idciclolectivo
      WHERE a.idalumno = ?
    `, [idalumno]);

    // 2. DATOS DE LA PERSONA
    const datosPersona = await this.personasService.getDatosPersona(datosCarrera.idpersona);

    // 3. DATOS COMPLEMENTARIOS
    const datosComplementarios = await
this.datosComplementariosPersonaService.getDatosComplementariosPersona(datosCarrera.idpersona);

    // 4. OBTENER TODOS LOS CATÁLOGOS NECESARIOS
    const [
      habitosEstudio,
      modalidadesEstudio,
      tiposConocimientosInformaticos,
      nivelesConocimientosInformaticos,
      actividadesTiempoLibre,
      nivelesIdioma
    ] = await Promise.all([
      this.datosService.getHabitosEstudio(),
      this.datosService.getModalidadesEstudio(),
      this.datosService.getTiposConocimientosInformaticos(),
      this.datosService.getNivelesConocimientosInformaticos(),
      this.datosService.getActividadesTiempoLibre(),
      this.datosService.getNivelesIdioma()
    ]);

    // 5. OBTENER DATOS DE LA SEDE
    let ciudadSede = 'Concepción del Uruguay'; // valor por defecto

    if (datosCarrera.idsedeinscripcion) {
      try {
        const datosSede = await this.datosService.getDatosSede(datosCarrera.idsedeinscripcion);
        if (datosSede && datosSede.ciudad) {
          ciudadSede = `${datosSede.provincia}, ${datosSede.ciudad}`;
        }
      } catch (error) {
        console.warn(`No se pudo obtener la ciudad de la sede ${datosCarrera.idsedeinscripcion}, usando valor por defecto`);
      }
    }
```

```
// 6. COMBINAR DATOS BÁSICOS
const data = {
  ...datosCarrera,
  ...datosPersona.datosGenerales,
  ...datosPersona.residenciaOrigen,
  ...datosPersona.residencia,
  ...datosComplementarios,
  header_title: 'UNIVERSIDAD DE CONCEPCIÓN DEL URUGUAY',
  header_subtitle: 'SOLICITUD DE INSCRIPCIÓN'
};

// 7. CALCULAR EDAD
if (data.fechanacimiento) {
  const calcularEdad = (fechaNac: string): number => {
    if (!fechaNac) return 0;

    const hoy = new Date();
    const nacimiento = new Date(fechaNac);
    let edad = hoy.getFullYear() - nacimiento.getFullYear();
    const mes = hoy.getMonth() - nacimiento.getMonth();

    if (mes < 0 || (mes == 0 && hoy.getDate() < nacimiento.getDate())) {
      edad--;
    }

    return edad;
  };

  data.edad = calcularEdad(data.fechanacimiento);
}

// 8. AGREGAR CIUDAD Y FECHA
if (!data.ciudad_fecha) {
  const fechaActual = new Date();
  const dia = fechaActual.getDate();
  const mes = fechaActual.toLocaleDateString('es-AR', { month: 'long' });
  const anio = fechaActual.getFullYear();
  data.ciudad_fecha = `${ciudadSede}, ${dia} de ${mes} de ${anio}`;
}

// 9. PREPARAR CONOCIMIENTOS INFORMÁTICOS
const conocimientosInformaticosCompletos = tiposConocimientosInformaticos.map(tipo => {
  const conocimientoAlumno = (data.conocimientosInformaticos || []).find(
    ci => ci.idtipoconocimientoinformatico == tipo.idtipoconocimientoinformatico
  );

  const niveles = nivelesConocimientosInformaticos.map(nivel => ({
    idnivelconocimientoinformatico: nivel.idnivelconocimientoinformatico,
    descripcion: nivel.descripcion,
    seleccionado: conocimientoAlumno?.idnivelconocimientoinformatico == nivel.idnivelconocimientoinformatico
  }));
```

```
  return {
    idtipoconocimientoinformatico: tipo.idtipoconocimientoinformatico,
    conocimiento: tipo.descripcion,
    niveles: niveles
  };
});

// 10. PREPARAR IDIOMAS
const idiomasCompletos = [];
if (data.idiomas && data.idiomas.length > 0) {
  data.idiomas.forEach(idioma => {
    const niveles = nivelesIdioma.map(nivel => ({
      idnivelidioma: nivel.idnivelidioma,
      descripcion: nivel.descripcion,
      seleccionado: idioma.idnivelidioma == nivel.idnivelidioma
    }));

    idiomasCompletos.push({
      idioma: idioma.idioma || idioma.descripcion || '',
      niveles: niveles
    });
  });
}

// 11. PREPARAR ACTIVIDADES TIEMPO LIBRE
const actividadesTiempoLibreCompletas = actividadesTiempoLibre.map(actividad => {
  const tieneActividad = (data.actividadesTiempoLibre || []).some(
    atl => atl.idactividadtiempolibre == actividad.idactividadtiempolibre
  );

  return {
    idactividadtiempolibre: actividad.idactividadtiempolibre,
    actividad: actividad.descripcion,
    seleccionada: tieneActividad
  };
});

// 12. BUSCAR ACTIVIDAD "OTROS"
let actividadOtraDescripcion = null;
if (data.actividadesTiempoLibre && data.actividadesTiempoLibre.length > 0) {
  const actividadOtros = data.actividadesTiempoLibre.find(
    atl => atl.idactividadtiempolibre == 9 && atl.descripcion_otra_actividad
  );

  if (actividadOtros && actividadOtros.descripcion_otra_actividad) {
    actividadOtraDescripcion = actividadOtros.descripcion_otra_actividad;
  }
}

// 13. PROCESAR PERSONAS CONVIVIENTES
const personasConvivientesProcesadas = (data.personasConvivientes || []).map(persona => {
  let nombreCompleto = '';
```

```
  if (persona.apellido && persona.nomfamiliar) {
    nombreCompleto = `${persona.apellido} ${persona.nomfamiliar}`;
  } else if (persona.nomfamiliar) {
    nombreCompleto = persona.nomfamiliar;
  } else if (persona.apellido) {
    nombreCompleto = persona.apellido;
  }

  return {
    ...persona,
    nombreCompleto: nombreCompleto.trim()
  };
});

// 14. PROCESAR DATOS PARENTALES
let datosMadre = null;
let datosPadre = null;

if (data.datosParentales && data.datosParentales.length > 0) {
  data.datosParentales.forEach(parental => {
    let apellido = '';
    let nombre = '';

    if (parental.nomfamiliar && !parental.apellido) {
      const partes = parental.nomfamiliar.trim().split(/\s+/);
      if (partes.length > 1) {
        apellido = partes[0];
        nombre = partes.slice(1).join(' ');
      } else {
        nombre = parental.nomfamiliar;
      }
    } else {
      apellido = parental.apellido || '';
      nombre = parental.nomfamiliar || '';
    }

    const parentalProcesado = {
      ...parental,
      apellido,
      nomfamiliar: nombre,
      nombreCompleto: `${apellido} ${nombre}`.trim()
    };

    const vinculoLower = (parental.vinculo || '').toLowerCase();

    if (vinculoLower == 'madre' || vinculoLower == 'mamá') {
      datosMadre = parentalProcesado;
    } else if (vinculoLower == 'padre' || vinculoLower == 'papá') {
      datosPadre = parentalProcesado;
    } else if (vinculoLower == 'tutor' || vinculoLower == 'tutora') {
      if (!datosMadre) {
        datosMadre = parentalProcesado;
      } else if (!datosPadre) {
```

```
          datosPadre = parentalProcesado;
        }
      }
    });
  }

  // 15. ARMAR EL OBJETO COMPLETO
  const dataCompleta = {
    ...data,

    nrodoc: formatearDNI(data.nrodoc),

    catalogos: {
      habitosEstudio,
      modalidadesEstudio,
      nivelesConocimientosInformaticos,
      actividadesTiempoLibre,
      nivelesIdioma
    },

    conocimientosInformaticosCompletos,
    actividadesTiempoLibreCompletas,
    actividadOtraDescripcion,
    idiomasCompletos,
    personasConvivientes: personasConvivientesProcesadas,

    // Datos parentales separados
    datosMadre,
    datosPadre,
    tieneDatosMadre: datosMadre != null,
    tieneDatosPadre: datosPadre != null,

    tieneOtrosEstudios: data.otrosEstudios && data.otrosEstudios.length > 0,
    tieneIdiomas: data.idiomas && data.idiomas.length > 0,

    idhabitoestudio_seleccionado: data.habitosEstudio?.idhabitoestudio || null,
    idmodalidadestudio_seleccionado: data.habitosEstudio?.idmodalidadestudio || null,
  };

  console.log('ESTOS SON LOS DATROS COMPLETOS DEL LOG DE UCUGESTION: ', dataCompleta);

  // 16. GENERAR PDF
  const registrarEnBD = true;
  const result = await this.pdfService.generateSolicitudInscripcion(
    dataCompleta,
    registrarEnBD
  );

  await queryRunner.commitTransaction();
  return result;

} catch (error) {
  await queryRunner.rollbackTransaction();
```

```
    throw error;
  } finally {
    await queryRunner.release();
  }
}
```