Módulo Convocatorias

- Lista de convocatorias internas (1) - [getInternalCalls]
- Obtiene combos (1) - [combosJobApplicant]
- Aplicar convocatoria (1) - [setJobApplicantPerApp]

Lista de convocatorias internas (1) - [getInternalCalls]

Descripción

*Obtiene todas las **convocatorias internas vigentes** dentro del ERP, incluyendo información del puesto, sede y su archivo adjunto (PDF u otro documento del Job Opening).*
El servicio cruza información entre:

- **Job Opening**
- **File** (archivos adjuntos al Job Opening)

Solo trae convocatorias internas **activas (status = "Open")** y marcadas como **convocatoria interna = 1**.

? Endpoint

GET /get-internal-calls

> No recibe parámetros.
> Toda la información se obtiene mediante apiService() hacia el ERP.

? Seguridad

Requiere conexión autorizada vía `apiService()`, usando token interno del ERP.

? Flujo del Servicio (Resumen)

1. **Obtener todos los Job Openings internos activos**

```
GET Job Opening?limit=None
    &fields=["name","designation","sucursal"]
    &filters=[
        ["convocatoria_interna","=",1],
        ["status","=","Open"]
    ]
```

2. **Obtener todos los archivos relacionados al Job Opening**

```
GET File?limit=None
    &fields=["name","attached_to_name","file_url"]
    &filters=[["attached_to_doctype","=","Job Opening"]]
```

3. Si alguno de los endpoints devuelve error, se retorna:

```
{ "valor": false, "msn": "Error al buscar las convocatorias internas." }
```

4. Si no hay convocatorias abiertas:

```
{ "valor": true, "msn": "En este momento no hay convocatorias internas..." }
```

5. Para cada convocatoria se agrega:
   - La descripción de la sede: `"PARA LA SEDE {sucursal}"`
   - El archivo correspondiente (`file_url`), si existe.

6. Se retorna listado de convocatorias internas enriquecidas.

? Request Body

No tiene body.

```
{}
```

? Response 200 – Ejemplo

```
{
  "valor": true,
  "msn": "Convocatorias internas encontradas",
  "data": [
    {
      "name": "JOB-0001",
      "designation": "Analista de Operaciones",
      "sucursal": "LIMA",
      "sede": "PARA LA SEDE LIMA",
      "file": "https://dominio.com/files/job_0001.pdf"
    }
  ]
}
```

? Posibles Errores

1. Error en consulta de Job Opening


```
{
  "valor": false,
  "msn": "Error al buscar las convocatorias internas.",
  "data": {}
}
```


2. Error en consulta de archivos File


```
{
  "valor": false,
  "msn": "Error al buscar las convocatorias internas.",
  "data": {}
}
```


3. No existen convocatorias internas activas


```
{
  "valor": true,
  "msn": "En este momento no hay convocatorias internas, Estar atento a las proximas convocatorias",
  "data": []
}
```

4. Error inesperado del servidor

```
{
  "valor": false,
  "msn": "Error del servidor",
  "data": "<error completo>"
}
```

? Esquemas (estructuras utilizadas)

Job Opening

Campos usados:

```
{
  "name": "string",
  "designation": "string",
  "sucursal": "string"
}
```

File

Campos usados:

```
{
  "name": "string",
  "attached_to_name": "string",
  "file_url": "string"
}
```

? Pseudo-código de la lógica

```
jobs = GET Job Opening internas y abiertas

if jobs vacío → return no hay convocatorias

files = GET File adjuntos a Job Opening

for job in jobs:
    job.sede = "PARA LA SEDE " + job.sucursal
    job.file = ""
    for file in files:
        if file.attached_to_name == job.name:
            job.file = BASE_CAPACITACION + file.file_url

return lista de jobs con su archivo si existe
```

Obtiene combos (1) - [combosJobApplicant]

□□ Descripción

*Este servicio obtiene y devuelve todos los combos (listas de valores) necesarios para el registro o postulación de un **Job Applicant** (Postulante) dentro del sistema.*

Los datos se obtienen directamente desde el ERP mediante el método interno `getCombo()`, que consulta tablas maestras como Country, Branch, Department y Gender.

Además, incluye una lista estática de tipos de documentos admitidos (DNI, Pasaporte, Carnet de Extranjería).

? Endpoint

GET /combos-job-applicant

No requiere parámetros.

? Seguridad

Requiere autenticación válida en la API del ERP, ya que las consultas internas se realizan mediante: `$this->getCombo("<Doctype>")`

? Flujo del Servicio (resumen real)

1. Inicializa un arreglo vacío `$combo`.
2. Obtiene desde el ERP las siguientes listas:
   - Países → Doctype: `Country`
   - Sucursales → Doctype: `Branch`

- Departamentos → Doctype: `Department`
- Género → Doctype: `Gender`

3. Agrega una lista fija de tipos de documentos:

   `["DNI", "Pasaporte", "Carnet de Extranjeria"]`
4. Retorna la estructura completa con todos los combos.
5. En caso de error, devuelve una respuesta controlada.

? Request Body

Este servicio **no recibe body**, ni parámetros.

`{}`

? Response 200 – Ejemplo

```
{
  "valor": true,
  "msn": "Combos generados correctamente",
  "data": {
    "paises": [
      { "name": "Peru" },
      { "name": "Chile" }
    ],
    "sucursales": [
      { "name": "Lima" }
    ],
    "departamento": [
      { "name": "Recursos Humanos" }
    ],
    "genero": [
      { "name": "Masculino" },
      { "name": "Femenino" }
    ],
    "documentos": [
      "DNI",
      "Pasaporte",
      "Carnet de Extranjeria"
    ]
  }
}
```

? Posibles Errores

1. Error de ejecución o conexión con el ERP

```
{
  "valor": false,
  "msn": "Error del servidor",
  "data": { "message": "detalle del error" }
}
```

? Schemas (estructuras usadas)

Resultado de getCombo()

El servicio asume que `getCombo()` devuelve una estructura similar a:

```
[
  { "name": "string" }
]
```

Lista fija de tipos de documentos:

```
[
  "DNI",
  "Pasaporte",
  "Carnet de Extranjeria"
]
```

? Lógica en pseudo-código

```
combo = {}

combo["paises"] = getCombo("Country")
combo["sucursales"] = getCombo("Branch")
combo["departamento"] = getCombo("Department")
combo["genero"] = getCombo("Gender")
combo["documentos"] = ["DNI", "Pasaporte", "Carnet de Extranjeria"]

return {
   valor: true,
   msn: "Combos generados correctamente",
   data: combo
}
```

Aplicar convocatoria (1) - [setJobApplicantPerApp]

? Descripción

Registra una nueva **solicitud de postulación (Job Applicant)** desde la aplicación móvil, validando previamente:

- Que el usuario autenticado no tenga ya una solicitud creada para el mismo puesto.
- Que exista un empleado vinculado al email de sesión.
- Que los campos obligatorios del formulario estén correctamente enviados.
- Que la información personal del empleado se complemente para generar el registro.

Este servicio crea un documento `Job Applicant` en el ERP y también registra un `Error Log` con la data enviada al ERP (para auditoría).

? Endpoint

POST /set-job-applicant-per-app

? Parámetros requeridos (Request Body)

```
{
  "email_logued": "string (obligatorio)",
  "job_title": "string (obligatorio)",
  "email_id": "string (obligatorio, debe contener @)",
  "direccion": "string",
  "adjuntar_copia_de_documento": "string (file url)",
  "sucursal": "string",
  "resume_attachment": "string (file url)",
  "grado_instruccion": "string",
  "phone_number": "string (opcional)",
  "country": "string (opcional)",
  "ciudad": "string (opcional)",
  "nacionalidad": "string (opcional)",
  "carrera_profesional": "string (opcional)"
}
```

? Seguridad

- Requiere usuario autenticado en la app.
- La comunicación con el ERP se realiza mediante `apiService()`.
- No requiere token específico del usuario final, es un servicio interno controlado.

? Flujo del Servicio (Explicación Detallada)

1?? Validaciones iniciales

- Verifica que exista `email_logued`.
- Verifica que el email del formulario (`email_id`) sea válido y contenga “@”.
- Valida campos obligatorios: dirección, documentos, sucursal, CV, grado de instrucción, job_title.

2?? Verifica si ya existe una postulación previa

Consulta en el ERP:

```
GET Job Applicant
?limit=None
&filters=[["email_logued","=","<email_logued>"], ["job_title","=","<job_title>"]]
```

Si ya existe un registro → **retorna error**.

3?? Obtiene datos del empleado

Busca el empleado vinculado al email logueado:

```
GET Employee
?fields=["*"]
&filters=[["user_id","=","<email_logued>"]]
```

Si no existe → **retorna error**.

4?? Calcula la edad del postulante

Usando la fecha de nacimiento del empleado mediante `DateTime::diff`.

5?? Construye el cuerpo del nuevo Job Applicant

Incluye:

- Datos del empleado (nombres, apellidos, género, fecha de nacimiento, DNI, nacionalidad).
- Datos enviados desde la app (CV, copia de documento, email, sucursal, carrera, etc.).
- Estado inicial del proceso: `"status": "Open"`
- Email logueado para relacionarlo a la cuenta APP.

6?? Crea el registro Job Applicant

```
POST Job Applicant
BODY: { ...data }
```

7?? Registra error interno en Error Log (auditoría)

La respuesta del ERP se envía también a: `POST Error Log`

8?? Retorna la respuesta al cliente

Si se creó correctamente:

```
{
  "valor": true,
  "msn": "Solicitud insertada correctamente",
  "data": { ... }
}
```

Si falló:

```
{
  "valor": false,
  "msn": "Solicitud no se puede insertar correctamente",
  "data": { ... }
}
```

? Respuesta 200 – Ejemplo (Éxito)


```
{
  "valor": true,
  "msn": "Solicitud insertada correctamente",
  "data": {
    "name": "JOB-APP-000123",
    "status": "Open",
    "applicant_name": "Juan Pérez",
    "job_title": "VAC-2025-005",
    "email_logued": "empleado@empresa.com"
  }
}
```


? Respuesta 200 – Ejemplo (Error)


```
{
  "valor": false,
  "msn": "Ya existe una solicitud para su usuario",
  "data": []
}
```

? Posibles Errores Comunes

1. Falta email_logued


```
{
  "valor": false,
  "msn": "Es necesario el email con el que ha entrado a la app"
}
```


2. Email sin “@”


```
{
  "valor": false,
  "msn": "El email debe contener @"
}
```


3. Empleado no existe

```
{
  "valor": false,
  "msn": "Error al traer el empleado asociado"
}
```

4. Solicitud duplicada

```
{
  "valor": false,
  "msn": "Ya existe una solicitud para su usuario"
}
```

5. Error al crear Job Applicant

```
{
  "valor": false,
  "msn": "Solicitud no se puede insertar correctamente",
  "data": { ...ERP response... }
}
```

? Estructuras usadas (Schemas)

? Employee (GET)

Campos consultados:

```
{
  "first_name": "string",
  "middle_name": "string",
  "first_last_name": "string",
  "second_last_name": "string",
  "nombre_completo": "string",
  "gender": "string",
  "passport_number": "string",
  "date_of_birth": "date",
  "place_of_issue": "string"
}
```

? Job Applicant (POST)

Campos enviados:

```
{
  "grado": "string",
  "job_title": "string",
  "applicant_name": "string",
  "numero_de_documento": "string",
  "email_id": "string",
  "sucursal": "string",
  "resume_attachment": "string",
  "status": "Open",
  "email_logued": "string"
}
```

? Pseudocódigo del Servicio

```
if !email_logued: error

// validar duplicados
list = GET Job Applicant by email_logued & job_title
if list > 0: error

employee = GET Employee by user_id=email_logued
if not exists: error

validar campos obligatorios
edad = calcular edad(employee.date_of_birth)

body = construir Job Applicant
createJob = POST Job Applicant

registrar Error Log con createJob

if createJob.data:
    return success
else:
    return error
```