Complemento de Paternidad API

API REST para calcular y gestionar el Complemento de Paternidad según la normativa española.

🚀 Características

- **Validación robusta** con Pydantic
- **Documentación automática** con OpenAPI/Swagger
- **Logging estructurado** en formato JSON
- **Tests comprehensivos** con cobertura >80%
- **Despliegue fácil** en Heroku
- **Cálculos precisos** según normativa oficial española

📋 Funcionalidades

Períodos de Aplicación

Período 1 (01/01/2016 - 03/02/2021)

- **Jubilaciones** (excepto anticipadas voluntarias)
- **Pensiones de viudedad**
- **Pensiones de incapacidad**
- **Requisito**: Mínimo 2 hijos
- **Cálculo porcentual** sobre la cuantía de la pensión:
    - 2 hijos → 5%
- 3 hijos → 10%
- ≥4 hijos → 15%

Período 2 (desde 04/02/2021)

- **Jubilaciones** (ordinarias y anticipadas)
- **Pensiones de incapacidad**
- **Pensiones de viudedad**
- **Importe fijo**: 35,90€ por hijo (límite: 4 hijos)
- Solo puede cobrarse uno de los dos posibles complementos (el de menor cuantía)

Endpoints Disponibles

`GET /eligibility`

Verificar si el solicitante cumple los criterios básicos.

Parámetros:

- `pension_type`: jubilacion|incapacidad|viudedad
- `start_date`: YYYY-MM-DD
- `num_children`: integer ≥ 1

`POST /calculate`

Calcular el complemento de paternidad.

Body JSON:

```
{
  "pension_type": "jubilacion",
  "start_date": "2021-06-15",
  "num_children": 2,
  "pension_amount": 1500.0
}
```

`GET /retroactive`

Calcular atrasos acumulados entre dos fechas.

Parámetros:

- `start_date`: YYYY-MM-DD
- `end_date`: YYYY-MM-DD
- `pension_amount`: float
- `num_children`: integer

`POST /compare`

Comparar dos progenitores para determinar quién tiene derecho.

Body JSON:

```
{
  "progenitor_1": {
    "name": "María",
    "pension_amount": 1000.0,
    "num_children": 2,
    "start_date": "2021-06-15",
    "pension_type": "jubilacion"
  },
  "progenitor_2": {
    "name": "José",
    "pension_amount": 1200.0,
    "num_children": 2,
    "start_date": "2021-06-15",
    "pension_type": "jubilacion"
  }
}
```

`GET /health`

Verificación de salud del servicio.

`GET /spec`

Especificación OpenAPI completa.

🛠️ Instalación y Desarrollo

Requisitos

- Python 3.10.8
- pip

Configuración Local

1. **Clonar el repositorio:**

```
git clone <repository-url>
cd complemento_api
```

2. **Crear entorno virtual:**

```
python -m venv venv
source venv/bin/activate  # En Windows: venv\Scripts\activate
```

3. **Instalar dependencias:**

```
pip install -r requirements.txt
```

4. **Ejecutar la aplicación:**

```
uvicorn app:app --reload --host 0.0.0.0 --port 8000
```

5. **Acceder a la documentación:**

- Swagger UI: http://localhost:8000/docs
- ReDoc: http://localhost:8000/redoc

Ejecutar Tests

```
# Ejecutar todos los tests
pytest

# Ejecutar con reporte de cobertura
pytest --cov=app --cov-report=html

# Ejecutar solo tests unitarios
pytest -m unit

# Ejecutar solo tests de integración
pytest -m integration
```

🚀 Despliegue en Heroku

Preparación

1. **Instalar Heroku CLI**
2. **Crear aplicación en Heroku:**

```
heroku create tu-app-complemento-paternidad
```

3. **Configurar variables de entorno:**

```
heroku config:set LOG_LEVEL=INFO
heroku config:set JSON_LOGS=true
```

Despliegue

```
# Hacer deploy
git push heroku main

# Verificar logs
heroku logs --tail

# Abrir aplicación
heroku open
```

Archivos de Configuración

- `Procfile`: Define el comando de inicio
- `runtime.txt`: Especifica la versión de Python
- `requirements.txt`: Dependencias del proyecto

📊 Estructura del Proyecto

```
complemento_api/
├── app/
│   ├── __init__.py          # Configuración de la aplicación FastAPI
│   ├── routes.py            # Definición de endpoints REST
│   ├── schemas.py           # Modelos Pydantic para validación
│   ├── services.py          # Lógica de negocio
│   ├── utils.py             # Funciones auxiliares
│   └── logging_config.py    # Configuración de logging
├── tests/
│   ├── __init__.py
│   ├── test_services.py     # Tests unitarios de servicios
│   ├── test_utils.py        # Tests de utilidades
│   └── test_api.py          # Tests de integración API
├── requirements.txt         # Dependencias Python
├── runtime.txt              # Versión de Python para Heroku
├── Procfile                 # Configuración de Heroku
├── pytest.ini              # Configuración de pytest
└── README.md                # Documentación
```

🔧 Configuración

Variables de Entorno

- `LOG_LEVEL`: Nivel de logging (DEBUG, INFO, WARNING, ERROR)
- `JSON_LOGS`: Activar logs en formato JSON (true/false)

Logging

El sistema de logging está configurado para generar logs estructurados en formato JSON, incluyendo:

- Timestamp UTC
- Nivel de log
- Módulo y función
- Mensaje
- Información adicional (request_id, duration, etc.)

📋 Ejemplos de Uso

Verificar Elegibilidad

```
curl "http://localhost:8000/eligibility?
pension_type=jubilacion&start_date=2021-06-15&num_children=2"
```

Calcular Complemento

```
curl -X POST "http://localhost:8000/calculate" \
  -H "Content-Type: application/json" \
  -d '{
    "pension_type": "jubilacion",
    "start_date": "2021-06-15",
    "num_children": 2,
    "pension_amount": 1500.0
  }'
```

Calcular Atrasos

```
curl "http://localhost:8000/retroactive?start_date=2021-05-01&end_date=2021-08-01&pension_amount=1000&num_children=2"
```

⚖️ Normativa Legal

Esta API implementa los cálculos según:

- **Real Decreto-ley 3/2021** de 2 de febrero
- **Ley 21/2021** de 28 de diciembre
- Normativa de la Seguridad Social española

🤝 Contribuir

1. Fork el proyecto
2. Crear una rama para la nueva característica
3. Commit los cambios
4. Push a la rama
5. Abrir un Pull Request

📄 Licencia

Este proyecto está bajo la Licencia MIT. Ver el archivo `LICENSE` para más detalles.

📞 Soporte

Para soporte técnico o preguntas sobre la implementación, por favor abrir un issue en el repositorio.