PRUEBA TÉCNICA .NET
Salutic Soluciones S.L.

3.2. Consultas

1. Listado de procesos de selección informando todos los campos

```
select process.*,process_status."status_name" from
salutic."process" as process
left join salutic."process_status" as process_status on
process."status_id" = process_status."id"
```

```
await DbContext.Processes
                .Include(p => p.ProcessStatus)
                .ToListAsync();
```

2. Listado de candidatos de un proceso de selección informando todos los campos

```
select applicant.* from salutic."applicant_process" as
applicant_process
right join salutic."applicant"  as applicant on applicant."id" =
applicant_process."applicant_id"
where applicant_process."process_id" = 1
```

```
await DbContext.Applicants
               .Include(p => p.ApplicantProcess)
                 .ThenInclude(ap => ap.ApplicantStatus)
               .Where(p => p.ApplicantProcess.ProcessId == 1)
               .ToListAsync();
```

3. Listado de candidatos que lleva un reclutador, indicando tanto su estado como el proceso de selección al cual se encuentra.

```
select applicant.*, process_status."status_name" as "process
stautus",applicant_status."status_name" as "Applicant status"
from salutic."applicant" as applicant
join salutic."applicant_process" as applicant_process on
applicant_process."applicant_id" = applicant."id"
join salutic."applicant_status" as applicant_status on
applicant_status."id" = applicant_process."applicant_status_id"
join salutic."recruiter_process" as recruiter_process on
recruiter_process."process_id" = applicant_process."process_id"
join salutic."process" as process on
recruiter_process."process_id" = process."id"
join salutic."process_status" as process_status on
process."status_id" = process_status."id"
join salutic."recruiter" as recruiter on
recruiter_process."recruiter_id" = recruiter."id"
where recruiter."username" = 'test'
```

```
var applicants = await DbContext.Applicants
                                .Where(a =>
a.ApplicantProcess.Process.RecruiterProcess.Recruiter.Username ==
"test").ToListAsync(); // Usando Lazy loading
```

4. Listado de candidatos, indicando para cada uno de ellos los procesos de selección en los que se encuentran.

```
select applicant.*,process."process_name" as "Process"
from salutic."applicant" as applicant
left join salutic."applicant_process" as applicant_process on
applicant_process."applicant_id" = applicant."id"
left join salutic."process" as process on
applicant_process."process_id" = process."id"
```

```
 await DbContext.ApplicantProcesses
                .Include(a => a.Applicant)
                .Include(a => a.Process)
              .ToListAsync();
```

Con **ApplicantProcesses** ya tendríamos todos los datos necesarios para listar la lista de usuarios, luego lo mapeamos con autoMapper y ya los tendriamos todos

5. Por anualidad, indicar el número de procesos de selección creados, en borrador, reclutando y finalizado.

```
select date_part('year', process."created_date") as
"Year", status."status_name" as "Process status",
count(process."id")
from salutic."process" as process
join salutic."process_status" as status on
process."status_id" = status."id"
Group by "Year", "Process status"
```

```
await DbContext.Processes
               .Include(p => p.ProcessStatus)
               .GroupBy(p => new { p.CreatedDate.Year,
p.ProcessStatus.StatusName })
               .Select(a => new { a.Key.Year, ProcessStatus =
a.Key.StatusName, Count = a.Count() })
                                    .ToListAsync();
```

6 - Esta pregunta está contestada junto con las consultas de SQL (Los segundos recuadros)

7. ¿Cómo inyectarías de forma masiva 100 candidatos en un proceso de selección?

Para inyectar de forma masiva 100 candidatos, lo que yo haría sería una paginación donde se crearían "batches" o paquetes de 10 elementos o del número de elementos que se quisieran insertar en los paquetes, porque de esta manera, estamos dividiendo la tarea en “x” partes con menos elementos y eso puede generar que el proceso, al meter un número reducido de elementos e invocarlo más veces, se ejecute más rápido que introduciendo 100 elementos de una sola vez.

4. Arquitectura

Para este caso pienso que la arquitectura más adecuada es una **arquitectura de microservicios** ya que los servicios que se piden **no requieren** de mucho **coste computacional** dado que son **funciones sencillas** y **rápidas** de implementar debido a que los datos manejados no requieren de funciones complejas cuyos algoritmos sean costosos de resolver.

Además de ser una arquitectura de microservicios. La he mezclado con una **arquitectura por capas**. Lo he mezclado para que así la API pueda ser más **modularizable** y **escalable,** ya que cada capa tiene una función concreta y trabaja de forma independiente.

5.1 Definición

Definida en el archivo “**Salutic API Definition**”

5.3 Entornos

Las variables que incluiría en dichos entornos serían las **diferentes conexiones** que la API tiene **según el entorno** donde se encuentre. Con esto quiero decir, podríamos tener varios entornos de desarrollo con bases de datos distintas, pongamos de ejemplo el entorno "Development", entonces, la conexión de base de datos, sería distinta posiblemente con un nombre de base de datos distinta y con un host distinto al de "Production" por ejemplo, que sería otro entorno.

Lo mismo pasaría con las **URL's de las posibles API's** que la nuestra pueda consumir, dependiendo si estamos consumiendo una api de prueba, puede tener una URL u otra, normalmente las URL de prueba no suelen tener los certificados SSL, serían del tipo HTTP.

Otra variable de entorno importante a incluir sería **ASPNETCORE_ENVIRONMENT**. Esta le indica a la API en qué entorno se encuentra y dependiendo del entorno, coge un archivo de configuración u otro, es decir, un appSettings.json distinto.

5.4 Patrones de diseño

Los patrones de diseño que he usado en la API han sido, **Dependency Injection** y el **patrón repository**, menciono este último ya que creo una capa independiente para la conexión con la base de datos y son los repositorios los encargados de hacer cualquier actividad relacionada con la base de datos